

















# **COMPUTER FUNDAMENTALS**

## **CONCEPTS, SYSTEMS & APPLICATIONS**







# **COMPUTER FUNDAMENTALS**

## **CONCEPTS, SYSTEMS & APPLICATIONS**

*BY*

**P.K. SINHA**  
UNIVERSITY OF TOKYO, JAPAN

**BPB PUBLICATIONS**  
B - 14, CONNAUGHT PLACE, NEW DELHI-110001



© BPB PUBLICATIONS, New Delhi

FIRST EDITION-1990

SECOND EDITION-1992

**Distributors:**

**BPB BOOK CENTRE**

376, Old Lajpat Rai Market, Delhi-110 006

**BUSINESS PROMOTION BUREAU**

8/1, Ritchie Street, Mount Road, Madras-600 002

**BUSINESS PROMOTION BUREAU**

4-3-268-C, Giriraj Lane, Bank Street, Hyderabad-500 195

**COMPUTER BOOK CENTRE**

12, Shrungar Complex, M. G. Road, Bangalore-560 001

**COMPUTER BOOK CENTRE**

Kothi No.-535, Sector-7, Panchkula-134 109, CHANDIGARH

**Notice**

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without the written permission of BPB Publications.



## ACKNOWLEDGEMENTS

The credit for the publication of the book goes to my father, Dr. Man Mohan Prasad, from whom I got inspiration to write it.

A book of this type naturally gained a number of ideas from previous books on this subject. The author expresses his thanks to all these authors, too numerous to acknowledge individually.

The author would also like to express his heart felt appreciation to his wife who drew some of the figures in the book and cheerfully devoted herself to the book-writing project.

MARCH, 1990

P.K. SINHA







## PREFACE

This book is intended for anyone who is interested in computers. It is designed to meet the needs of beginners as well as advanced learners who wish to deal with computers. As computers are being used today in every walk of life, this book can be used by students of computer science, business administration, management, engineering, and general science as well. At the undergraduate level, where an introductory course on computers is included as a part of the curriculum, this book can be used as a textbook. The book can also serve as course material for participants of short-term course conducted by various training organizations. Programmers and other computer professionals will also find the book helpful and informative. In short, this book is for everyone who is either excited about computers or interested in knowing about computers.

A knowledge of computers is not a prerequisite to follow the contents of the book. It assumes no background in computers or computer programming. Starting with an introductory chapter, the book goes on to cover all facets of digital computers. It uses the simple-to-complex and easy-to-learn approach throughout. Special attention has been paid to remove unwanted details and to make the material comprehensive, concise and practical. Simple and plain English has been used throughout and the use of computer jargons has been avoided as far as practicable.

All the concepts presented in the book are illustrated with suitable examples as and when required. This will enable the readers to use the book effectively for self-study. In addition to this, each chapter contains a number of review questions and exercises which are carefully designed to provide an insight into the subject matter.

MARCH, 1990

P.K. SINHA







# CONTENTS

|                                                                 |    |
|-----------------------------------------------------------------|----|
| <b>1. INTRODUCTION</b>                                          | 1  |
| CHARACTERISITCS OF COMPUTERS                                    | 1  |
| THE EVOLUTION OF COMPUTERS                                      | 2  |
| -- The Mark I Computer                                          |    |
| -- The Atanasoff - Berry Computer                               |    |
| -- The ENIAC                                                    |    |
| -- The EDVAC                                                    |    |
| -- The EDSAC                                                    |    |
| -- Manchester Mark I                                            |    |
| -- The UNIVAC I                                                 | 4  |
| THE COMPUTER GENERATIONS                                        |    |
| -- First Generation                                             |    |
| -- Second Generation                                            |    |
| -- Third Generation                                             |    |
| -- Fourth Generation                                            |    |
| -- Fifth Generation                                             |    |
| <b>2. BASIC COMPUTER ORGANIZATION</b>                           | 9  |
| INPUT UNIT                                                      | 10 |
| OUTPUT UNIT                                                     | 10 |
| STORAGE UNIT                                                    | 10 |
| ARITHMETIC LOGIC UNIT                                           | 11 |
| CONTROL UNIT                                                    | 11 |
| CENTRAL PROCESSING UNIT                                         | 11 |
| THE SYSTEM CONCEPT                                              | 13 |
| <b>3. NUMBER SYSTEMS</b>                                        | 13 |
| NON-POSITIONAL NUMBER SYSTEMS                                   | 13 |
| POSITIONAL NUMBER SYSTEMS                                       | 13 |
| -- Binary Number System                                         |    |
| -- Octal Number System                                          |    |
| -- Hexadecimal Number system                                    | 15 |
| CONVERTING FROM ONE NUMBER SYSTEM TO ANOTHER                    |    |
| -- Converting To Decimal From Another Base                      |    |
| -- Converting From Base 10 To A New Base                        |    |
| -- Converting From A Base Other Than 10 To A Base Other Than 10 |    |
| -- Shortcut Method For Binary To Octal Conversion               |    |
| -- Shortcut Method For Octal To Binary Conversion               |    |
| -- Shortcut Method For Binary To Hexadecimal Conversion         |    |
| -- Shortcut Method For Hexadecimal To Binary Conversion         |    |
| FRACTIONAL NUMBERS                                              | 22 |



|                                              |    |
|----------------------------------------------|----|
| <b>4. COMPUTER CODES</b>                     | 25 |
| BCD CODE                                     | 25 |
| EBCDIC                                       | 27 |
| -- Zoned and Packed Decimal Numbers          |    |
| ASCII                                        | 29 |
| COLLATING SEQUENCE                           | 31 |
| <b>5. COMPUTER ARITHMETIC</b>                | 35 |
| WHY BINARY                                   | 35 |
| BINARY ARITHMETIC                            | 36 |
| -- Addition                                  |    |
| -- Substraction                              |    |
| -- Multiplication                            |    |
| -- Division                                  |    |
| <b>6. BOOLEAN ALGEBRA AND LOGIC CIRCUITS</b> | 45 |
| BOOLEAN ALGEBRA                              | 45 |
| -- Fundamental Concepts of Boolean Algebra   |    |
| -- Postulates of Boolean Algebra             |    |
| -- The Principle of Duality                  |    |
| -- Theorems of Boolean Algebra               |    |
| BOOLEAN FUNCTIONS                            | 51 |
| -- Minimization of Boolean Functions         |    |
| -- Complement of a Function                  |    |
| -- Canonical Forms For Boolean Functions     |    |
| -- Conversion Between Canonical Forms        |    |
| LOGIC GATES                                  | 57 |
| -- AND GATE                                  |    |
| -- OR GATE                                   |    |
| -- NOT GATE                                  |    |
| -- NAND GATE                                 |    |
| -- NOR GATE                                  |    |
| LOGIC CIRCUITS                               | 61 |
| -- Converting Expressions To Logic Circuits  |    |
| -- The Universal NAND Gate                   |    |
| -- The Universal NOR Gate                    |    |
| -- EXCLUSIVE-OR AND EQUIVALENCE Functions    |    |
| DESIGN OF COMBINATIONAL CIRCUITS             | 68 |
| -- Design Of Half-Adder                      |    |
| -- Design Of Full-Adder                      |    |
| -- A Parallel Binary Adder                   |    |
| <b>7. PRIMARY STORAGE</b>                    | 75 |
| STORAGE LOCATIONS AND ADDRESSES              | 75 |
| STORAGE CAPACITY                             | 76 |
| -- Why More Bits                             |    |
| -- Fixed And Variable Word Length Storage    |    |
| RAM, ROM, PROM AND EPROM                     | 77 |
| CACHE MEMORY                                 | 78 |
| REGISTERS                                    | 78 |



|                                            |            |
|--------------------------------------------|------------|
| <b>8. SECONDARY STORAGE DEVICES</b>        | <b>81</b>  |
| SEQUENTIAL AND DIRECT-ACCESS DEVICES       | 81         |
| PUNCHED PAPER TAPE                         | 82         |
| MAGNETIC TAPE                              | 82         |
| TAPE CASSETTES AND CARTRIDGES              | 84         |
| MAGNETIC DISK                              | 85         |
| FLOPPY DISK                                | 88         |
| WINCHESTER DISK                            | 89         |
| MAGNETIC DRUM                              | 89         |
| MASS STORAGE                               | 90         |
| OPTICAL DISK                               | 90         |
| MAGNETIC BUBBLE MEMORY                     | 91         |
| CHARGE-COUPLED DEVICE                      | 91         |
| A NOTE ON STORAGE HIERARCHY                | 92         |
| <b>9. INPUT - OUTPUT DEVICES</b>           | <b>95</b>  |
| PUNCHED HOLE DEVICES                       | 95         |
| -- Punched Cards                           |            |
| -- Punched Paper Tape                      |            |
| MAGNETIC MEDIA DEVICES                     | 98         |
| PRINTERS                                   | 98         |
| -- Character Printers                      |            |
| -- Line Printers                           |            |
| -- Page Printers                           |            |
| KEYBOARD DEVICES                           | 101        |
| -- Video Display Terminals                 |            |
| -- Teleprinter Terminals                   |            |
| -- Point-of-Sale Terminals                 |            |
| -- Intelligent Terminals                   |            |
| SCANNERS                                   | 104        |
| -- Optical Scanners                        |            |
| -- Magnetic-Ink Character Recognition      |            |
| OTHER DEVICES                              | 107        |
| -- Computer Output Microfilm               |            |
| -- Digitizers                              |            |
| -- Plotters                                |            |
| -- Voice Recognition and Response Devices  |            |
| OFFLINE DATA ENTRY DEVICES                 | 109        |
| -- Key-to-Card                             |            |
| -- Key-to-Tape                             |            |
| -- Key-to-Floppy                           |            |
| -- Key-to-Disk                             |            |
| <b>10. COMPUTER SOFTWARE</b>               | <b>113</b> |
| WHAT IS SOFTWARE                           | 113        |
| RELATIONSHIP BETWEEN HARDWARE AND SOFTWARE | 113        |
| TYPES OF SOFTWARE                          | 114        |
| ACQUIRING SOFTWARE                         | 115        |
| FIRMWARE                                   | 115        |



|                                                |     |
|------------------------------------------------|-----|
| <b>11. PLANNING THE COMPUTER PROGRAM</b>       | 117 |
| PURPOSE OF PROGRAM PLANNING                    | 117 |
| ALGORITHM                                      | 118 |
| FLOWCHARTS                                     | 118 |
| -- Flowchart Symbols                           |     |
| -- Sample Flowcharts                           |     |
| -- Levels of Flowcharts                        |     |
| -- Flowcharting Rules                          |     |
| -- Advantages of Flowcharts                    |     |
| -- Limitations of Flowcharts                   |     |
| DECISION TABLES                                | 130 |
| PSEUDOCODE                                     | 132 |
| <b>12. COMPUTER LANGUAGES</b>                  | 141 |
| ANALOGY WITH NATURAL LANGUAGES                 | 141 |
| MACHINE LANGUAGE                               | 142 |
| ASSEMBLY LANGUAGE                              | 143 |
| HIGH-LEVEL LANGUAGE                            | 148 |
| COMPILERS                                      | 149 |
| INTERPRETERS                                   | 151 |
| SOME HIGH LEVEL LANGUAGES                      | 152 |
| -- FORTRAN                                     |     |
| -- COBOL                                       |     |
| -- BASIC                                       |     |
| -- PASCAL                                      |     |
| -- PL/I                                        |     |
| -- Other High Level Languages                  |     |
| CHARACTERISTICS OF A GOOD LANGUAGE             | 158 |
| SUBROUTINES                                    | 159 |
| <b>13. SYSTEM IMPLEMENTATION AND OPERATION</b> | 161 |
| TESTING AND DEBUGGING                          | 161 |
| DOCUMENTATION                                  | 163 |
| CHANGEOVER TO THE NEW SYSTEM                   | 165 |
| -- Immediate Changeover                        |     |
| -- Parallel Run                                |     |
| -- Phased Conversion                           |     |
| SYSTEM EVALUATION                              | 167 |
| SYSTEM MAINTENANCE                             | 168 |
| <b>14. OPERATING SYSTEMS</b>                   | 171 |
| DEFINITION AND FUNCTIONS                       | 171 |
| EVOLUTION OF OPERATING SYSTEMS                 | 173 |
| BATCH PROCESSING                               | 173 |
| JOB CONTROL LANGUAGE                           | 174 |
| SPOOLING                                       | 175 |
| MULTIPROGRAMMING                               | 176 |
| MULTIPROCESSING                                | 178 |
| TIME-SHARING                                   | 180 |
| ON-LINE PROCESSING                             | 184 |



|                         |     |
|-------------------------|-----|
| REAL-TIME PROCESSING    | 184 |
| VIRTUAL STORAGE         | 185 |
| OS-CONTROLLED SOFTWARE  | 186 |
| -- Translating Programs |     |
| -- Library Programs     |     |
| -- Utility programs     |     |

## 15. BUSINESS DATA PROCESSING CONCEPTS 189

|                                                    |     |
|----------------------------------------------------|-----|
| WHAT IS DATA PROCESSING                            | 189 |
| DATA STORAGE HIERARCHY                             | 189 |
| FILE ORGANIZATIONS                                 | 190 |
| -- Sequential Files                                |     |
| -- Direct Files                                    |     |
| -- Indexed Sequential Files                        |     |
| FILE UTILITIES                                     | 193 |
| -- Sorting                                         |     |
| -- Searching                                       |     |
| -- Merging                                         |     |
| -- Copying                                         |     |
| -- Printing                                        |     |
| -- Maintenance                                     |     |
| -- Labelling                                       |     |
| -- Scratching                                      |     |
| DATA BASE SYSTEMS                                  | 195 |
| -- The Data Base Concept                           |     |
| -- Data Base Management System                     |     |
| -- Data Base Structuring Techniques                |     |
| -- Advantages and Limitations of Data Base Systems |     |

## 16. DATA COMMUNICATIONS AND COMPUTER NETWORKS 203

|                                          |     |
|------------------------------------------|-----|
| BASIC ELEMENTS OF A COMMUNICATION SYSTEM | 204 |
| DATA TRANSMISSION MODES                  | 205 |
| -- Simplex                               |     |
| -- Half-duplex                           |     |
| -- Full-duplex                           |     |
| DATA TRANSMISSION SPEED                  | 205 |
| -- Narrowband                            |     |
| -- Voiceband                             |     |
| -- Broadband                             |     |
| TRANSMISSION MEDIA                       | 205 |
| -- Wire pairs                            |     |
| -- Coaxial cable                         |     |
| -- Microwave system                      |     |
| -- Communications satellite              |     |
| -- Optical fibers                        |     |
| DIGITAL: AND ANALOG TRANSMISSION         | 208 |
| -- Amplitude Modulation                  |     |
| -- Frequency Modulation                  |     |
| -- Phase Modulation                      |     |
| COMMUNICATIONS PROCESSORS                | 210 |
| -- Multiplexers                          |     |
| -- Concentrators                         |     |



|                                                |     |
|------------------------------------------------|-----|
| -- Front End Processors                        |     |
| ASYNCHRONOUS AND SYNCHRONOUS TRANSMISSION      | 213 |
| SWITCHING TECHNIQUES                           | 214 |
| -- Circuit Switching                           |     |
| -- Message Switching                           |     |
| -- Packet Switching                            |     |
| NETWORK TOPOLOGIES                             | 217 |
| -- Star Network                                |     |
| -- Ring Network                                |     |
| -- Completely Connected Network                |     |
| -- Hybrid Network                              |     |
| -- Multipoint Network                          |     |
| LOCAL AREA NETWORK AND WIDE AREA NETWORK       | 220 |
| ✓ COMMUNICATION PROTOCOLS                      | 221 |
| ✓ -- Roles of Protocol                         |     |
| -- The OSI Model                               |     |
| -- Concept of Layering in OSI Model            |     |
| -- An Example of Message Transfer in OSI Model |     |
| DISTRIBUTED DATA PROCESSING                    | 225 |
| GLOSSARY                                       | 229 |



## CHAPTER 1

# 1. INTRODUCTION

The word "computer" comes from the word "compute" which means to calculate. So a computer is normally considered to be a calculating device that can perform arithmetic operations at enormous speed.

In fact, the original objective for inventing the computer was to create a fast calculating machine. But more than 80% of the work done by computers today is of non-mathematical or non-numerical nature. Hence, to define a computer merely as calculating device is to ignore over 80% of its work.

More accurately, a computer may be defined as a device that operates upon information or data. Data can be anything like bio-data of various applicants when the computer is used for recruiting personnel, or the marks obtained by various students in various subjects when the computer is used to prepare results, or the details (name, age, sex, etc.) of various passengers when the computer is employed for making airline or railway reservations, or numbers of different types in case of application of computers for scientific research problems, etc.

Thus, data comes in various shapes and sizes

depending upon the type of computer application. A computer can store, process, and retrieve data as and when desired. The fact that computers process data is so fundamental that many people have started calling it a *data processor*.

The name data processor is more inclusive because modern computers not only compute in the usual sense but also perform other functions with the data that flow to and from them. For example, data processors may gather data from various incoming sources, merge (process of mixing or putting together) them all, sort (process of arranging in some sequence - ascending or descending) them in the desired order and finally print them in the desired format. None of these operations involve the arithmetic operations normally associated with a computing device but the term computer is often applied anyway.

## CHARACTERISTICS OF COMPUTERS

1. **Speed.** A computer is a very fast device. It can perform in a few seconds the amount of work that a human being can do in an entire year - if he worked day and night and did nothing else. To put it in a different manner, a



computer does in one minute what would take a man his entire lifetime.

While talking about the speed of a computer, we do not talk in terms of seconds or even milliseconds ( $10^{-3}$ ). Our units of speed are the microseconds ( $10^{-6}$ ), the nanoseconds ( $10^{-9}$ ), and even the picoseconds ( $10^{-12}$ ). A powerful computer is capable of performing about 3 to 4 million simple arithmetic operations per second.

**2. Accuracy.** The accuracy of a computer is consistently high and the degree of accuracy of a particular computer depends upon its design. But for a particular computer, each and every calculation is performed with the same accuracy.

Errors can occur in a computer, but these are mainly due to human rather than technological weaknesses, that is, due to imprecise thinking by the programmer (a person who writes instructions for a computer to solve a particular problem) or due to inaccurate data.

**3. Diligence.** Unlike human beings, a computer is free from monotony, tiredness, lack of concentration, etc., and hence can work for hours together without creating any error and without grumbling. Due to this property, computers obviously score over human beings in doing routine type of jobs which require great accuracy. If ten million calculations have to be performed, a computer will perform the ten millionth calculation with exactly the same accuracy and speed as the first one.

**4. Versatility.** Versatility is one of the most wonderful things about the computer. One moment, it is preparing the results of particular examination, the next moment it is busy preparing electricity bills, and in between, it may be helping an office secretary to trace an important letter in seconds. All that is required to change its talent is to slip in a new program (a sequence of instructions for the computer) into it. Briefly, a computer is capable of performing almost any task provided that the task can be reduced to a series of logical steps.

**5. Power of Remembering.** As a human being acquires new knowledge, the brain subconsciously selects what it feels to be important and worth retaining in its memory, and relegates unimportant details to the back of the mind or just forgets them. With computers, this is not the case. A computer can store and recall any amount of information because of its secondary storage (a type of detachable memory) capability. Every piece of information can be retained as long as desired by the user and can be recalled as and when required. Even after several years, the information recalled will be as accurate as on the day when it was fed to the computer. A computer forgets or loses

certain information only when it is asked to do so. So it is entirely upto the user to make a computer retain or forget a particular information.

**6. No I.Q.** A computer is not a magical device. It can only perform tasks that a human being can. The difference is that it performs these tasks with unthinkable speed and accuracy. It possesses no intelligence of its own. Its I.Q. is zero, at least till today. It has to be told what to do and in what sequence. Hence, only the user can determine what tasks a computer will perform. A computer cannot take its own decision in this regard.

**7. No Feelings.** Computers are devoid of emotions. They have no feelings and no instincts because they are machine. Although men have succeeded in building a memory for the computer, but no computer possesses the equivalent of a human heart and soul. Based on our feelings, taste, knowledge, and experience, we often make certain judgements in our day to day life. But computers cannot make such judgements on their own. Their judgement is based on the instructions given to them in the form of programs that are written by us. They are only as good as man makes and uses them.

## THE EVOLUTION OF COMPUTERS

Necessity is the mother of invention. The saying holds true for computers also because computers were invented as a result of man's search for fast and accurate calculating devices.

The earliest device that qualifies as a digital computer is the "abacus" also known as "soroban". This device permits the users to represent numbers by the position of beads on a rack. Simple addition and subtraction can be carried out rapidly and efficiently by positioning the beads appropriately. Although, the abacus was invented around 600 B.C., it is interesting to note that it is still used in the Far East and its users can calculate at amazing speeds.

Another manual calculating device was John Napier's bone or cardboard multiplication calculator. It was designed in the early 17th century and its upgraded versions were in use even around 1890.

The first mechanical adding machine was invented by Blaise Pascal in 1642. Later, in the year 1671, Baron Gottfried Wilhelm von Leibniz of Germany invented the first calculator for multiplication. Keyboard machines originated in the United States around 1880 and are extensively used even today. Around this period only, Herman Hollerith came up with the concept of punched



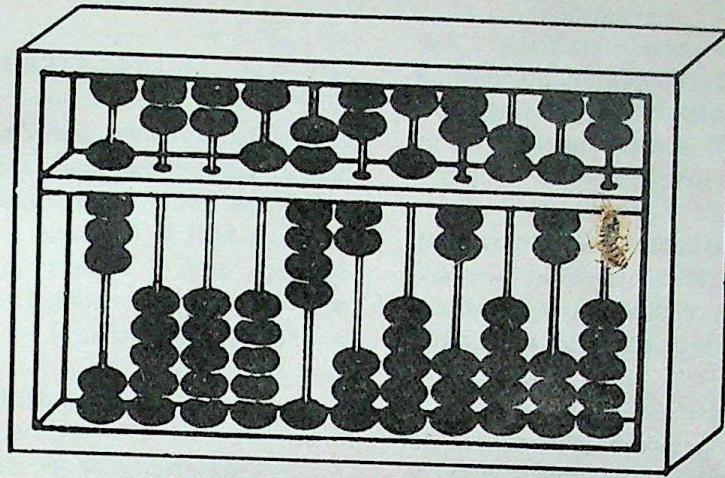


Figure 1.1. A thirteenth century abacus.

cards which are extensively used as input media in modern digital computers. Business machines and calculators made their appearance in Europe and America towards the end of the nineteenth century.

Charles Babbage, a nineteenth century Professor at Cambridge University, is considered to be the father of modern digital computers. During his period, mathematical and statistical tables were prepared by a group of clerks. Even the utmost care and precautions could not eliminate human errors. Babbage had to spend several hours checking these tables. Soon he became dissatisfied and exasperated with this type of monotonous job. The result was that he started thinking to build a machine which could compute tables guaranteed to be error-free. In this process, Babbage designed a "Difference Engine" in the year 1822 which could produce reliable tables. In 1842, Babbage came out with his new idea of Analytical Engine that was intended to be completely automatic. It was to be capable of performing the basic arithmetic functions for any mathematical problem and it was to do so at an average speed of 60 additions per minute. Unfortunately, he was unable to produce a working model of this machine mainly because the precision engineering required to manufacture the machine was not available during that period. However, his efforts established a number of principles which have been shown to be fundamental to the design of any digital computer. We will now discuss about some of the well known early computers.

#### THE MARK I COMPUTER (1937-44)

Also known as Automatic Sequence Controlled

calculator, this was the first fully automatic calculating machine designed by Howard A. Aiken of Harvard University in collaboration with IBM (International Business Machines) corporation. Its design was based on the techniques already developed for punched card machinery.

Although this machine proved to be extremely reliable, it was very complex in design and huge in size. It used over 3000 electrically actuated switches to control its operations and was approximately 50 feet long and 8 feet high. It was capable of performing five basic arithmetic operations : addition, subtraction, multiplication, division, and table reference. A number as big as 23 decimal digits could be used in this machine. It took approximately 0.3 second to add two numbers and 4.5 seconds for multiplication of two numbers. Hence, the machine was very slow as compared to today's computers.

It was basically an electro-mechanical device since both mechanical and electronic components were used in its design. Although its operations were not controlled electronically, Aiken's machine is often classified as computer because its instructions, which were entered by means of punched paper tape, could be altered.

#### THE ATANASOFF - BERRY COMPUTER (1939-42)

This electronic machine was developed by Dr. John Atanasoff to solve certain mathematical equations. It was called the Atanasoff - Berry Computer, or ABC, after its inventor's name and his assistant, Clifford Berry. It used 45 vacuum tubes for internal logic and capacitors for storage.



## THE ENIAC (1943-46)

The Electronic Numerical Integrator And Calculator (ENIAC) was the first all electronic computer. It was constructed at the Moore School of Engineering of the University of Pennsylvania, U.S.A by a design team led by Professors J. Presper Eckert and John Mauchly.

ENIAC was developed as a result of military need. It took up the wall space in a 20 X 40 square feet room and used 18,000 vacuum tubes. The addition of two numbers was achieved in 200 microseconds, and multiplication in 2000 microseconds.

Although, much faster in speed as compared to Mark I computer, ENIAC had two major shortcomings : it could store and manipulate only a very limited amount of information, and its programs were wired on boards. These limitations made it difficult to detect errors and to change the programs. Hence its use was limited. However, whatever be the shortcomings of ENIAC, it represented an impressive feat of electronic engineering and was used for many years to solve ballistic problems.

## THE EDVAC (1946-52)

The operation of ENIAC was seriously handicapped by the wiring board. This problem was later overcome by the new concept of "stored program" developed by Dr. John Von Neumann. The basic idea behind the stored program concept is that a sequence of instructions as well as data can be stored in the memory of the computer for the purpose of automatically directing the flow of operations. The stored program feature considerably influenced the development of modern digital computers and because of this feature we often refer to modern digital computers as stored program digital computers. The Electronic Discrete Variable Automatic Computer (EDVAC) was designed on stored program concept. Von Neumann has also got a share of the credit for introducing the idea of storing both instructions and data in the binary form ( a system that uses only two digits - 0 & 1 to represent all characters ) instead of the decimal numbers or human readable words.

## THE EDSAC (1947-49)

Almost simultaneously with EDVAC of U.S.A., the Britishers developed the Electronic Delay Storage Automatic Calculator (EDSAC). The machine executed its first program in May 1949. In this machine, addition operation was accomplished in 1500 microseconds, and multiplication operation in 4000 microseconds. The machine was developed by a group of scientists headed by Professor Maurice Wilkes at the Cambridge University.

## Mathematical Laboratory.

## MANCHESTER MARK I (1948)

This computer was a small experimental machine based on the stored program concept. It was designed at Manchester University by a group of scientists headed by Professor M.H.A. Newman. Its storage capacity was only 32 words, each of 31 binary digits. This was too limited to store data and instructions. Hence, the Manchester Mark I was hardly of any practical use.

## THE UNIVAC I (1951)

The Universal Automatic Computer (UNIVAC) was the first digital computer which was not "one of a kind". Many UNIVAC machines were produced, the first of which was installed in the Census Bureau in 1951 and was used continuously for 10 years. The first business use of a computer, a UNIVAC I, was by General Electric Corporation in 1954.

In 1952, the International Business Machines (IBM) Corporation introduced the 701 commercial computer. In rapid succession, improved models of the UNIVAC I and other 700-series machines were introduced. In 1953, IBM produced the IBM-650 and sold over 1000 of these computers.

The commercially available digital computers, that could be used for business and scientific applications, had arrived.

## THE COMPUTER GENERATIONS

"Generation" in computer talk is a step in technology. It provides a framework for the growth of the computer industry. Originally, the term 'generation' was used to distinguish between varying hardware technologies. But nowadays, it has been extended to include both the hardware and the software (see Chapter 10 for definition of hardware and software) which together make up an entire computer system.

The custom of referring to the computer era in terms of generations came into wide use only after 1964. There are totally five computer generations known till today. Each generation has been discussed below in detail along with their advantages and disadvantages. Although there is a certain amount of overlapping between the generations, the approximate dates shown against each are normally accepted.



## FIRST GENERATION (1942-1955)

We have already discussed about some of the early computers - ENIAC, EDVAC, EDSAC, etc. These machines and other of their time were made possible by the invention of "vacuum tube", which was a fragile glass device that could control and amplify electronic signals. These vacuum tube computers are referred to as first-generation computers.

### Advantages

1. Vacuum tubes were the only electronic components available during those days.
2. Vacuum tube technology made possible the advent of electronic digital computers.
3. These computers were the fastest calculating device of their time. They could perform computations in milliseconds.

### Disadvantages

1. Too bulky in size
2. Unreliable
3. Thousands of vacuum tubes that were used emitted large amount of heat and burnt out frequently
4. Air conditioning required
5. Prone to frequent hardware failures
6. Constant maintenance required
7. Nonportable
8. Manual assembly of individual components into functioning unit required
9. Commercial production was difficult and costly
10. Limited commercial use

## SECOND GENERATION (1955-1964)

The transistor, a smaller and more reliable successor to the vacuum tube, was invented in 1947. However, computers that used transistors were not produced in quantity until over a decade later. The second generation emerged with transistors being the brain of the computer.

With both the first and the second generation computers, the basic component was a discrete or separate entity. The many thousands of individual components had to be assembled by hand into functioning circuits. The manual assembly of individual components and the cost of labour involved at this assembly stage made the commercial production of these computers difficult and costly.

### Advantages

1. Smaller in size as compared to first generation computers
2. More reliable
3. Less heat generated
4. These computers were able to reduce computational times from milliseconds to microseconds.
5. Less prone to hardware failures
6. Better portability
7. Wider commercial use

### Disadvantages

1. Air-conditioning required
2. Frequent maintenance required
3. Manual assembly of individual components into a functioning unit was required
4. Commercial production was difficult and costly.

## THIRD GENERATION (1964-1975)

Advances in electronics technology continued and the advent of "microelectronics" technology made it possible to integrate large number of circuit elements into very small (less than 5 mm square) surface of silicon known as "chips". This new technology was called "integrated circuits" (ICs). The third generation was based on IC technology and the computers that were designed with the use of integrated circuits were called third generation computers.

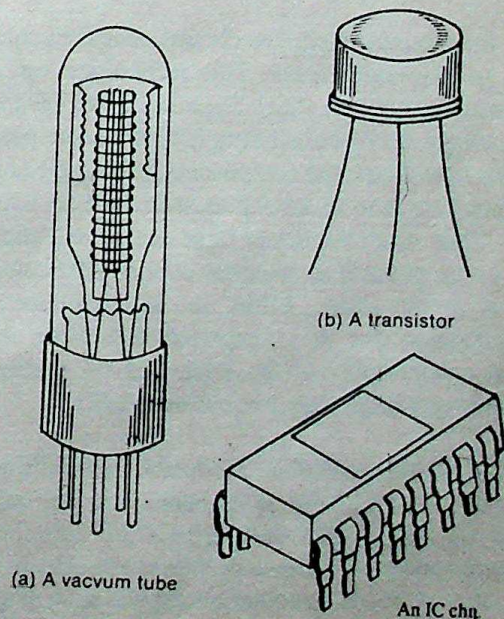


Figure 1.2. Electronic devices used for manufacturing computers of different generations.



**Advantages**

1. Smaller in size as compared to previous generation computers.
2. Even more reliable than second generation computers.
3. Even lower heat generated than second generation computers.
4. These computers were able to reduce computational times from microseconds to nanoseconds.
5. Maintenance cost is low because hardware failures are rare.
6. Easily portable.
7. Totally general purpose. Widely used for various commercial applications all over the world.
8. Less power requirement than previous generation computers.
9. Manual assembly of individual components into a functioning unit not required. So human labour and cost involved at assembly stage reduced drastically.
10. Commercial production was easier and cheaper.

**Disadvantages**

1. Air-conditioning required in many cases.
2. Highly sophisticated technology required for the manufacture of IC chips.

**FOURTH GENERATION (1975 ONWARDS)**

Initially, the integrated circuits contained only about ten to twenty components. This technology was named small scale integration (SSI). Later, with the advancement in technology for manufacturing ICs, it became possible to integrate upto a hundred components on a single chip. This technology came to be known as medium scale integration (MSI). Then came the era of large scale integration (LSI) when it was possible to integrate over 30,000 components onto a single chip. Effort is still on for further miniaturization and it is expected that more than one million components will be integrated on a single chip known as very large scale integration (VLSI).

A fourth generation computer, which is what we have now, has LSI chips as its brain. It is LSI technology which has led to the development of very small but extremely powerful computers. It was the start of a social revolution. A whole computer circuit was soon available on a single chip, the size of a postage stamp. Overnight computers became incredibly compact. They became inexpensive to make and suddenly it became possible for anyone and every one to own a computer.

**Advantages**

1. Smallest in size because of high component density
2. Very reliable
3. Heat generated is negligible
4. No air conditioning required in most cases
5. Much faster in computation than previous generations
6. Hardware failure is negligible and hence minimal maintenance is required
7. Easily portable because of their small size
8. Totally general purpose
9. Minimal labour and cost involved at assembly stage
10. Cheapest among all generations

**Disadvantage**

1. Highly sophisticated technology required for the manufacture of LSI chips.

**FIFTH GENERATION (YET TO COME)**

Scientists are now at work on the fifth generation computers - a promise, but not yet a reality. They aim to bring us machines with genuine I.Q., the ability to reason logically, and with real knowledge of the world. Thus, unlike the last four generations which naturally followed its predecessor, the fifth generation will be totally different, totally novel, totally new.

In structure it will be parallel (the present ones are serial) and will be able to do multiple tasks simultaneously. In functions, it will not be algorithmic (step by step, with one step at a time). In nature, it will not do just data processing (number crunching) but knowledge processing. In inference, it will not be merely deductive, but also inductive. In application, it will behave like an expert. In programming, it will interact with humans in ordinary language (unlike BASIC, COBOL, FORTRAN, etc. which present computers need). And in architecture, it will have KIPS (Knowledge Information Processing System) rather than the present DIPS/LIPS (Data/Logic Information Processing System).

The odds of coming out with a fifth generation computer are heaviest for Japan. They have already started work in this direction few years back. Japan has chosen the PROLOG (Programming in Logic) language as its operating software and plans to have the final machine talk with human beings, see and deliver pictures and hear the normal, natural language.



## QUESTIONS

1. What is a computer ? Why is it also known as a data processor ?
2. List out and explain some of the important characteristics of a computer.
3. What is an abacus ?
4. Who is known as the father of modern digital computers and why ?
5. Why are modern digital computers often referred to as stored program digital computers ?
6. Give the full form of the following abbreviations used in computer terminology :  
IBM, ENIAC, EDVAC, EDSAC, UNIVAC.
7. What is meant by 'generation' in computer terminology ? How many computer generations are there till now ?
8. List out the various computer generations along with their basic characteristics.
9. Write a short note on fifth generation computers.
10. What are the advantages of transistors over vacuum tubes ?
11. What is an IC ? How does it help in reducing the size of computers ?
12. List out some of the advantages of IC technology over transistor technology.
13. Give the full form of the following abbreviations used in computer terminology :  
IC, SSI, MSI, LSI, VLSI, DIPS, LIPS, PROLOG.







## CHAPTER 2

# 2. BASIC COMPUTER ORGANIZATION

All computer systems perform the following five basic operations :

1. **Inputting.** The process of entering data and instructions into the computer system.
2. **Storing.** Saving data and instructions so that they are available for initial or for additional processing as and when required.
3. **Processing.** Performing arithmetic operations or logical operations (comparisons like equal to, less than, greater than, etc.) on data in order to convert them into useful information.
4. **Outputting.** The process of producing useful information or results for the user, such as a printed report or visual display.

5. **Controlling.** Directing the manner and sequence in which all of the above operations are performed.

The goal of this chapter is to familiarize you with the computer system units that perform these functions. This chapter will provide you with an overview of computer systems as they are viewed by computer system architects. It is an introduction to chapters 7, 8 and 9 which describe the major units and their functions in more detail.

The internal architectural design of computers differs from one system model to another. However, the basic organization remains the same for all computer systems. A block diagram of the basic computer organization is shown in Figure 2.1. In this figure, the solid lines are used to indicate the flow of instruction and data, and the dotted lines represent the control exercised by the control unit. It displays the five major building blocks, or



functional units, of a digital computer system. These five units correspond to the five basic operations performed by all computer systems. The function of each of these units is described below:

## INPUT UNIT

Data and instructions must enter the computer system before any computation can be performed on the supplied data. This task is performed by the input unit that links the external environment with the computer system. Data and instructions enter input units in forms that depend upon the particular device used. For example, data is entered from a keyboard in a manner similar to typing, and this differs from the way in which data is entered through a card reader which is another type of input device. However, regardless of the form in which they receive their inputs, all input devices must provide a computer with data that are transformed into the binary codes that the primary memory of a computer is designed to accept. This transformation is accomplished by units called *input interfaces*. Input interfaces are designed to match the unique physical or electrical characteristics of input devices to the requirements of the computer system.

In short, the following functions are performed by an input unit :

1. It accepts (or reads) the list of instructions and data from the outside world.
2. It converts these instructions and data in computer acceptable form.
3. It supplies the converted instructions and data to the computer system for further processing.

## OUTPUT UNIT

The job of an output unit is just the reverse of that of an input unit. It supplies information and results of computation to the outside world. Thus it links the computer with the external environment. As computers work with binary code, the results produced are also in the binary form. Hence, before supplying the results to the outside world, it must be converted to human acceptable (readable) form. This task is accomplished by units called *output interfaces*. Output interfaces are designed to match the unique physical or electrical characteristics of output devices (terminals, printers, etc.) to the requirements of the external environment.

In short, the following functions are performed by an output unit :

1. It accepts the results produced by the computer which are in coded form and hence cannot be easily understood by us.
2. It converts these coded results to human acceptable (readable) form.
3. It supplies the converted results to the outside world.

## STORAGE UNIT

The data and instructions that are entered into the computer system through input units have to be stored inside the computer before the actual processing starts. Similarly, the results produced by the computer after processing must also be kept somewhere inside the computer system before being passed on to the output units. Moreover, the intermediate results produced by the computer must also be preserved for ongoing processing. The storage unit or the primary/main storage of a computer system is designed to cater to all these needs. It provides space for storing data and instructions, space for intermediate results, and also space for the final results.

✓ In short, the specific functions of the storage unit are to hold (store) :

1. All the data to be processed and the instructions required for processing (received from input devices).
2. Intermediate results of processing.
3. Final results of processing before these results are released to an output device.

## ARITHMETIC LOGIC UNIT

The arithmetic logic unit (ALU) of a computer system is the place where the actual execution of the instructions takes place during the processing operation. To be more precise, all calculations are performed and all comparisons (decisions) are made in the ALU. The data and instructions, stored in the primary storage prior to processing, are transferred as and when needed to the ALU where processing takes place. No processing is done in the primary storage unit. Intermediate results generated in the ALU are temporarily transferred back to the primary storage until needed at a later time. Data may thus move from primary storage to ALU and back again to storage many times before the processing is over. After the completion of processing, the final results which are stored in the storage unit are released to an output device.

The type and number of arithmetic and logic operations that a computer can perform is determined by the engineering design of the ALU. (However, almost all



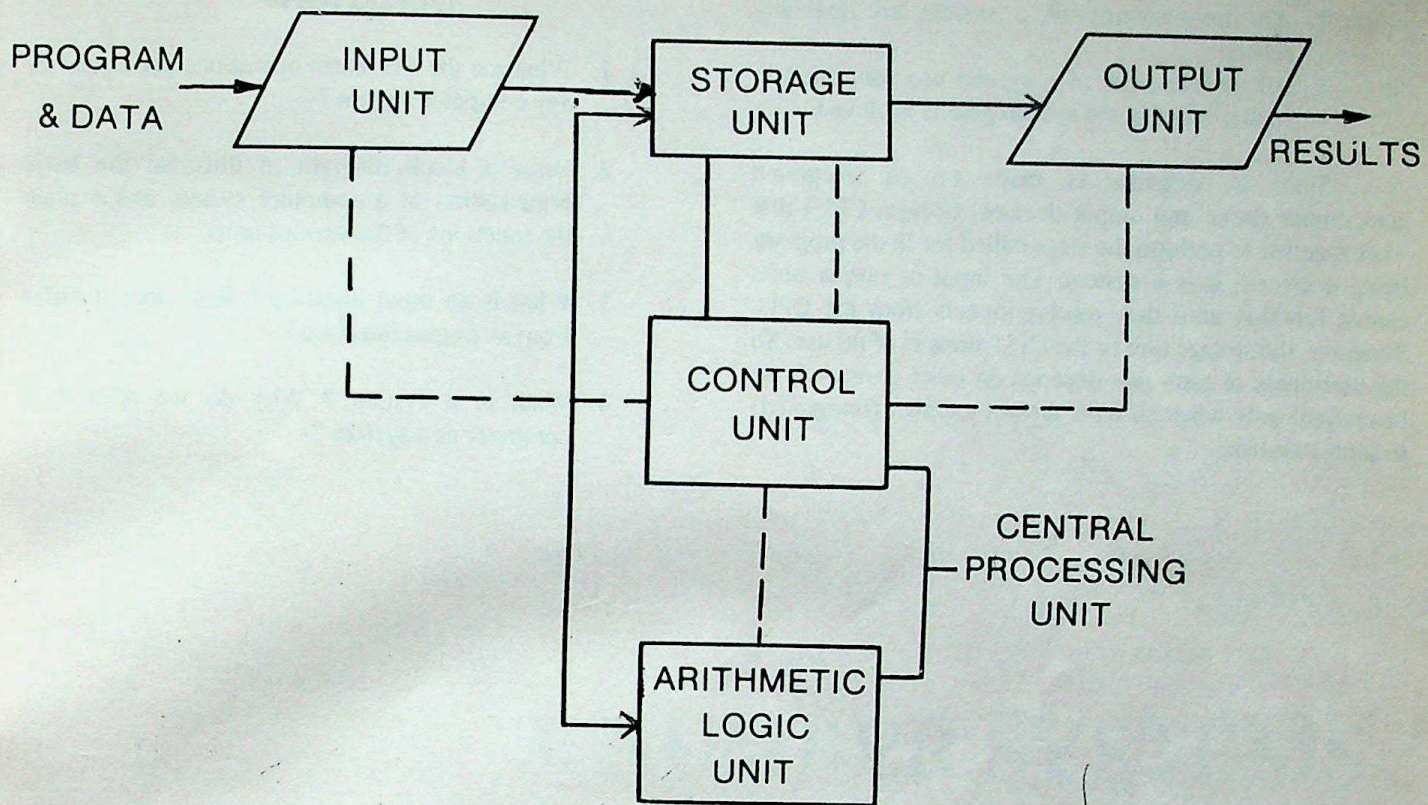


Figure 2.1. Basic organisation of a computer system

ALU's are designed to perform the four basic arithmetic operations - add, subtract, multiply, divide and logic operations or comparisons such as less than, equal to, or greater than.)

## CONTROL UNIT

How does the input device know that it is time for it to feed data into the storage unit? How does the ALU know what should be done with the data once they are received?

And how is it that only the final results are sent to the output device and not the intermediate results? All this is possible because of the control unit of the computer system. By selecting, interpreting, and seeing to the execution of the program instructions, the control unit is able to maintain order and direct the operation of the entire system. Although, it does not perform any actual processing on the data, the control unit acts as a central nervous system for the other components of the computer. (It manages and coordinates the entire computer system.) (It obtains instructions from the program stored in main memory, interprets the instructions, and issues signals that cause other units of the system to execute them.)

## CENTRAL PROCESSING UNIT

The control unit and the arithmetic logic unit of a computer system are jointly known as the Central Processing Unit (CPU). The CPU is the brain of any computer system. In a human body, all major decisions are taken by the brain and the other parts of the body function as directed by the brain. Similarly, in a computer system, all major calculations and comparisons are made inside the CPU and the CPU is also responsible for activating and controlling the operations of other units of a computer system.

## THE SYSTEM CONCEPT

You might have observed by now that we have been referring to a computer as a system (computer system). What can be the reason behind this? To know the answer, let us first consider the definition of a system.

A *system* is a group of integrated parts that have the common purpose of achieving some objective(s). So, the following three characteristics are key to a system:

1. A system has more than one element.



2. All the elements of a system are logically related.
3. All the elements of a system are controlled in such a way that the system goal is achieved.

Since a computer is made up of integrated components (input and output devices, storage, CPU) that work together to perform the steps called for in the program being executed, it is a system. The input or output units cannot function until they receive signals from the CPU. Similarly, the storage unit or the CPU alone is of no use. So the usefulness of each unit depends on other units and can be realized only when all units are put together (integrated) to form a system.

## QUESTIONS

1. What are the five basic operations performed by any computer system ?
2. Draw a block diagram to illustrate the basic organisation of a computer system and explain the functions of the various units.
3. What is an input interface ? How does it differ from an output interface ?
4. What is a system ? Why do we refer to a computer as a system ?



## CHAPTER 3

# 3. NUMBER SYSTEMS

We have already seen in the previous chapter that inside a computer system, data is stored in a format that cannot be easily read by human beings. This is the reason why input and output (I/O) interfaces are required. Every computer stores numbers, letters, and other special characters in a coded form. Before going into the details of these codes, it is essential to have a basic understanding of the number system. So the goal of this chapter is to familiarize you with the basic fundamentals of number system. It also introduces some of the commonly used number systems by computer professionals and the relationship between them.

Number systems are basically of two types: non-positional and positional.

### NON-POSITIONAL NUMBER SYSTEMS

In early days, human beings counted on fingers. When ten fingers were not adequate, stones, pebbles, or sticks were used to indicate values. This method of counting uses an additive approach or the non-positional number system. In this system, we have symbols such as I for 1, II for 2, III for 3, IV for 4, V for 5, etc. Each

symbol represents the same value regardless of its position in the number and the symbols are simply added to find out the value of a particular number. Since it is very difficult to perform arithmetic with such a number system, positional number systems were developed as the centuries passed.

### POSITIONAL NUMBER SYSTEMS

In a positional number system, there are only a few symbols called digits, and these symbols represent different values depending on the position they occupy in the number. The value of each digit in such a number is determined by three considerations :

1. the digit itself,
2. the position of the digit in the number, and
3. the base of the number system (where base is defined as the total number of digits available in the number system).

The number system that we use in our day-to-day life is called the *Decimal number system*. In this system, the base is equal to 10 because there are altogether ten symbols or digits (0,1,2,3,4,5,6,7,8,9) used in this system. You know that in the decimal system, the successive positions to the



left of the decimal point represent units, tens, hundreds, thousands, etc. But you may not have given much attention to the fact that each position represents a specific power of the base ( $10^n$ ). For example, the decimal number 2586 (written as  $2586_{10}$ ) consists of the digit 6 in the units position, 8 in the tens position, 5 in the hundreds position, and 2 in the thousands position and its value can be written as :

$$(2 \times 1000) + (5 \times 100) + (8 \times 10) + (6 \times 1)$$

$$\text{or } 2000 + 500 + 80 + 6$$

$$\text{or } 2586$$

It may also be observed that the same digit signifies different values depending on the position it occupies in the number. For example,

In  $2586_{10}$  the digit 6 signifies  $6 \times 10^0 = 6$

In  $2568_{10}$  the digit 6 signifies  $6 \times 10^1 = 60$

In  $2658_{10}$  the digit 6 signifies  $6 \times 10^2 = 600$

In  $6258_{10}$  the digit 6 signifies  $6 \times 10^3 = 6000$

Thus any number can be represented by using the available digits and arranging them in various positions.

The principles that apply to the decimal system apply in any other positional number system. It is important only to keep track of the base of the number system in which we are working.

There are two characteristics of all number systems that are suggested by the value of the base. In all number systems, the value of the base determines the total number of different symbols or digits available in the number system. The first of these choices is always zero. The second characteristic is that the maximum value of a single digit is always equal to one less than the value of the base.

Some of the number systems commonly used in computer design and by computer professionals are discussed below.

## BINARY NUMBER SYSTEM

The binary number system is exactly like the decimal system except that the base is 2 instead of 10. We have only two symbols or digits (0 and 1) that can be used in this number system. Note that the largest single digit is (one less than the base). Again, each position in a binary

number represents a power of the base (2). As such, in this system, the rightmost position is the units ( $2^0$ ) position, the second position from the right is the 2's ( $2^1$ ) position and proceeding in this way we have 4's ( $2^2$ ) position, 8's ( $2^3$ ) position, 16's ( $2^4$ ) position, and so on. Thus, the decimal equivalent of the binary number 10101 (written as  $10101_2$ ) is

$$(1 \times 2^4) + (0 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0)$$

$$\text{or } 16 + 0 + 4 + 0 + 1$$

$$\text{or } 21$$

In order to be specific about which system we are referring to, it is common practice to indicate the base as a subscript. Thus we write :

$$10101_2 = 21_{10}$$

"Binary digit" is often referred to by the common abbreviation *bit*. Thus, a "bit" in computer terminology means either a 0 or a 1. A binary number consisting of  $n$  bits is called an  $n$ -bit number. Table 3.1 lists all the 3-bit numbers along with their decimal equivalent. Remember that we have only two digits, 0 and 1, in the binary system, and hence the binary equivalent of the decimal number 2 has to be stated as 10 (read as one, zero). Another important point to note is that with 3 bits (positions), only 8 ( $2^3$ ) different patterns of 0's and 1's are possible and from Table 3.1 it may be seen that a 3-bit number can have one of the 8 values in the range 0 to 7. In fact, it can be shown that any decimal number in the range 0 to  $2^n - 1$  can be represented in the binary form as an  $n$ -bit number.

Every computer stores numbers, letters, and other special characters in binary form. There are several occasions when computer professionals have to know the raw data contained in a computer's memory. A common way of looking at the contents of a computer's memory is to print out the memory contents on the line printer. This print out is called a *memory dump*. If memory dumps were to be printed using binary numbers, the computer professionals would be confronted with many pages of 0s and 1s. Working with these numbers would be very difficult and error prone.

Because of the quantity of printout that would be required in a memory dump of binary digits and the lack of digit variety (0s and 1s only), two number systems, *octal* and *hexadecimal*, are used as shortcut notation for binary. These number systems and their relationship with the binary number system will now be explained in this chapter.



Table 3.1. 3-bit Numbers With Their Decimal Values.

| Binary | Decimal Equivalent |
|--------|--------------------|
| 000    | 0                  |
| 001    | 1                  |
| 010    | 2                  |
| 011    | 3                  |
| 100    | 4                  |
| 101    | 5                  |
| 110    | 6                  |
| 111    | 7                  |

### OCTAL NUMBER SYSTEM

In the octal number system the base is 8. So in this system there are only eight symbols or digits : 0,1,2,3,4,5,6 and 7 (8 and 9 do not exist in this system). Here also the largest single digit is 7 (one less than the base). Again, each position in an octal number represents a power of the base (8). Thus the decimal equivalent of the octal number 2057 (written as  $2057_8$ ) is :

$$(2 \times 8^3) + (0 \times 8^2) + (5 \times 8^1) + (7 \times 8^0)$$

$$\text{or } 1024 + 0 + 40 + 7$$

$$\text{or } 1071$$

$$\text{So we have } 2057_8 = 1071_{10}.$$

Observe that since there are only 8 digits in the octal number system, so 3 bits ( $2^3 = 8$ ) are sufficient to represent any octal number in binary (see Table 3.1).

### HEXADECIMAL NUMBER SYSTEM

The hexadecimal number system is one with a base of 16. The base of 16 suggests choices of 16 single-character digits or symbols. The first 10 digits are the digits of a decimal system 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. The remaining six digits are denoted by A, B, C, D, E, F representing the decimal values 10, 11, 12, 13, 14, 15 respectively. In the hexadecimal number system, therefore, the letters A through F are number digits. The number A has a decimal equivalent value of 10 and the hexadecimal F has a decimal equivalent value of 15. Thus, largest single digit is F or 15 (one less than the base). Again, each position in a hexadecimal system represents a power of the base (16). Thus the decimal equivalent of the hexadecimal number 1AF (written as  $1AF_{16}$ ) is:

$$(1 \times 16^2) + (A \times 16^1) + (F \times 16^0)$$

$$\text{or } (1 \times 256) + (10 \times 16) + (15 \times 1)$$

$$\text{or } 256 + 160 + 15$$

$$\text{or } 431$$

$$\text{Thus } 1AF_{16} = 431_{10}$$

Observe that since there are only 16 digits in the hexadecimal number system, so 4 bits ( $2^4 = 16$ ) are sufficient to represent any hexadecimal number in binary.

### CONVERTING FROM ONE NUMBER SYSTEM TO ANOTHER

Numbers expressed in decimal are much more meaningful to us than are values expressed in any other number system. This is mainly because of the fact that we have been using decimal numbers in our day-to-day life right from childhood. However, any number value in one number system can be represented in any other number system. Because the input and the final output values are to be in decimal, computer professionals are often required to convert numbers in other number systems to decimal and vice-versa. There are many methods or techniques that can be used to convert numbers from one base to another. We will see one technique used in converting to base 10 from any other base and a second technique to be used in converting from base 10 to any other base.

### CONVERTING TO DECIMAL FROM ANOTHER BASE

The following three steps are used to convert to a base 10 value from any other number system :



Step 1 : Determine the column (positional) value of each digit (this depends on the position of the digit and the base of the number system).

Step 2 : Multiply the obtained column values (in Step 1) by the digits in the corresponding columns.

Step 3 : Sum the products calculated in Step 2. The total is the equivalent value in decimal.

Example 3.1.  $11001_2 = ?_{10}$

Solution :

Step 1 : Determine column values

| Column Number<br>(from right) | Column Value |
|-------------------------------|--------------|
|-------------------------------|--------------|

|   |            |
|---|------------|
| 1 | $2^0 = 1$  |
| 2 | $2^1 = 2$  |
| 3 | $2^2 = 4$  |
| 4 | $2^3 = 8$  |
| 5 | $2^4 = 16$ |

Step 2 : Multiply column values by corresponding column digits

$$\begin{array}{r}
 16 \ 8 \ 4 \ 2 \ 1 \\
 \times 1 \ \times 1 \ \times 0 \ \times 0 \ \times 1 \\
 \hline
 16 \ 8 \ 0 \ 0 \ 1
 \end{array}$$

Step 3 : Sum the products

$$16 + 8 + 0 + 0 + 1 = 25$$

Hence,  $11001_2 = 25_{10}$

Example 3.2.  $4706_8 = ?_{10}$

Solution :

Step 1 :  
Column Number  
(from right)

Column Value

|   |             |
|---|-------------|
| 1 | $8^0 = 1$   |
| 2 | $8^1 = 8$   |
| 3 | $8^2 = 64$  |
| 4 | $8^3 = 512$ |

Step 2 :

$$\begin{array}{r}
 512 \quad 64 \quad 8 \quad 1 \\
 \times 4 \quad \times 7 \quad \times 0 \quad \times 6 \\
 \hline
 2048 \quad 448 \quad 0 \quad 6
 \end{array}$$

Step 3 :

$$2048 + 448 + 0 + 6 = 2502$$

$$\text{Hence, } 4706_8 = 2502_{10}$$

Example 3.3.  $1AC_{16} = ?_{10}$

Solution :

$$\begin{aligned}
 1AC_{16} &= 1 \times 16^2 + A \times 16^1 + C \times 16^0 \\
 &= 1 \times 256 + 10 \times 16 + 12 \times 1 \\
 &= 256 + 160 + 12 \\
 &= 428_{10}
 \end{aligned}$$

Example 3.4.  $4052_7 = ?_{10}$

Solution :

$$\begin{aligned}
 4052_7 &= 4 \times 7^3 + 0 \times 7^2 + 5 \times 7^1 + 2 \times 7^0 \\
 &= 4 \times 343 + 0 \times 49 + 5 \times 7 + 2 \times 1 \\
 &= 1372 + 0 + 35 + 2 \\
 &= 1409_{10}
 \end{aligned}$$

Example 3.5.  $4052_6 = ?_{10}$

Solution :

$$\begin{aligned}
 4052_6 &= 4 \times 6^3 + 0 \times 6^2 + 5 \times 6^1 + 2 \times 6^0 \\
 &= 4 \times 216 + 0 \times 36 + 5 \times 6 + 2 \times 1 \\
 &= 864 + 0 + 30 + 2 \\
 &= 896_{10}
 \end{aligned}$$



Comparing this result with the result obtained in Example 3.4, we find that although the digits (4052) are same for both the numbers, but their decimal equivalent is different. This is because of the fact that the number in Example 3.4 is represented in base 7 number system whereas the number in Example 3.5 is represented in base 6 system.

**Example 3.6.**  $11001_4 = ?_{10}$

**Solution :**

$$\begin{aligned} 11001_4 &= 1 \times 4^4 + 1 \times 4^3 + 0 \times 4^2 + 0 \times 4^1 + 1 \times 4^0 \\ &= 1 \times 256 + 1 \times 64 + 0 \times 16 + 0 \times 4 + 1 \times 1 \\ &= 256 + 64 + 0 + 0 + 1 \\ &= 321_{10} \end{aligned}$$

Compare the result with Example 3.1.

**Example 3.7.**  $1AC_{13} = ?_{10}$

**Solution :**

$$\begin{aligned} 1AC_{13} &= 1 \times 13^2 + A \times 13^1 + C \times 13^0 \\ &= 1 \times 169 + 10 \times 13 + 12 \times 1 \\ &= 311_{10} \end{aligned}$$

Compare the result with Example 3.3.

#### CONVERTING FROM BASE 10 TO A NEW BASE (DIVISION - REMAINDER TECHNIQUE)

The following four steps are used to convert a number from base 10 to a new base:

- Step 1 :** Divide the decimal number to be converted by the value of the new base.
- Step 2 :** Record the remainder from step 1 as the rightmost digit (least significant digit) of the new base number.
- Step 3 :** Divide the quotient of the previous divide by the new base.
- Step 4 :** Record the remainder from step 3 as the next digit (to the left) of the new base number.

Repeat steps 3 and 4, recording remainders from right to left, until the quotient becomes zero in step 3. Note that the last remainder thus obtained will be the most

significant digit (MSD) of the new base number.

**Example 3.8.**  $25_{10} = ?_2$

**Solution :**

- Steps 1 & 2:  $25/2 = 12$  and remainder 1
- Steps 3 & 4:  $12/2 = 6$  and remainder 0
- Steps 3 & 4:  $6/2 = 3$  and remainder 0
- Steps 3 & 4:  $3/2 = 1$  and remainder 1
- Steps 3 & 4:  $1/2 = 0$  and remainder 1

As mentioned in Steps 2 & 4, the remainders have to be arranged in the reverse order so that the first remainder becomes the least significant digit (LSD) and the last remainder becomes the most significant digit (MSD).

Hence,  $25_{10} = 11001_2$

Compare the result with Example 3.1.

**Example 3.9.**  $42_{10} = ?_2$

**Solution :**

| 2 |       | Remainders |
|---|-------|------------|
|   | 42    |            |
|   | ----- |            |
|   | 21    | 0          |
|   | ----- |            |
|   | 10    | 1          |
|   | ----- |            |
|   | 5     | 0          |
|   | ----- |            |
|   | 2     | 1          |
|   | ----- |            |
|   | 1     | 0          |
|   | ----- |            |
|   | 0     | 1          |

Hence,  $42_{10} = 101010_2$

**Example 3.10.**  $952_{10} = ?_8$



Solution :

| 8     | 952 | Remainders |
|-------|-----|------------|
| ----- |     |            |
|       | 119 | 0          |
| ----- |     |            |
|       | 14  | 7          |
| ----- |     |            |
|       | 1   | 6          |
| ----- |     |            |
|       | 0   | 1          |

Hence,  $952_{10} = 1670_8$

Example 3.11.  $428_{10} = ?_{16}$

Solution :

| 16    | 428 | Remainders in hexadecimal. |
|-------|-----|----------------------------|
| ----- |     |                            |
|       | 26  | $12 = C$                   |
| ----- |     |                            |
|       | 1   | $10 = A$                   |
| ----- |     |                            |
|       | 0   | $1 = 1$                    |

Hence,  $428_{10} = 1AC_{16}$

Compare the result with Example 3.3.

Example 3.12.  $100_{10} = ?_5$

Solution :

| 5     | 100 | Remainders |
|-------|-----|------------|
| ----- |     |            |
|       | 20  | 0          |
| ----- |     |            |
|       | 4   | 0          |
| ----- |     |            |
|       | 0   | 4          |

Hence,  $100_{10} = 400_5$

Example 3.13.  $100_{10} = ?_4$

Solution :

| 4     | 100 | Remainders |
|-------|-----|------------|
| ----- |     |            |
|       | 24  | 4          |
| ----- |     |            |
|       | 4   | 0          |
| ----- |     |            |
|       | 0   | 0          |

|       |    |   |
|-------|----|---|
| ----- | 25 | 0 |
|       | 6  | 1 |
| ----- |    |   |
|       | 1  | 2 |
| ----- |    |   |
|       | 0  | 1 |

Hence,  $100_{10} = 1210_4$

Compare this result with the result obtained in Example 3.12.

Example 3.14.  $1715_{10} = ?_{12}$

Solution:

| 12    | 1715 | Remainders in Base 12 |
|-------|------|-----------------------|
| ----- |      |                       |
|       | 142  | $11 = B$              |
| ----- |      |                       |
|       | 11   | $10 = A$              |
| ----- |      |                       |
|       | 0    | $11 = B$              |

Hence,  $1715_{10} = BAB_{12}$

### CONVERTING FROM A BASE OTHER THAN 10 TO A BASE OTHER THAN 10

The following two steps are used to convert a number from a base other than 10 to a base other than 10.

Step 1: Convert the original number to a decimal number (base 10).

Step 2: Convert the decimal number obtained to the new base.

Example 3.15.  $545_6 = ?_4$

Solution :

Step 1: Convert from base 6 to base 10  
 $545 = 5 \times 6^2 + 4 \times 6^1 + 5 \times 6^0$   
 $= 5 \times 36 + 4 \times 6 + 5 \times 1$   
 $= 180 + 24 + 5$



$$= 209_{10}$$

$$11010011_2 = 1 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0$$

Step 2 : Convert  $209_{10}$  to base 4

| 4     | 209 | Remainders |
|-------|-----|------------|
| ----- |     |            |
|       | 52  | 1          |
| ----- |     |            |
|       | 13  | 0          |
| ----- |     |            |
|       | 3   | 1          |
| ----- |     |            |
|       | 0   | 3          |

$$= 1 \times 128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 0 \times 4 + 1 \times 2 + 1 \times 1$$

$$= 128 + 64 + 0 + 16 + 0 + 0 + 2 + 1$$

$$= 211_{10}$$

Step 2 : Convert  $211_{10}$  to base 16

| 16    | 211 | Remainders            |
|-------|-----|-----------------------|
| ----- |     |                       |
|       | 13  | 3 = 3 in Hexadecimal  |
| ----- |     |                       |
|       | 0   | 13 = D in Hexadecimal |

Hence,  $209_{10} = 3101_4$

So,  $545_6 = 209_{10} = 3101_4$

Thus,  $545_6 = 3101_4$

Example 3.16.  $101110_2 = ?_8$

Solution :

Step 1 : Convert  $101110_2$  to base 10

$$\begin{aligned} 101110_2 &= 1 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 32 + 0 + 8 + 4 + 2 + 0 \\ &= 46_{10} \end{aligned}$$

Step 2 : Convert  $46_{10}$  to base 8.

| 8     | 46 | Remainders |
|-------|----|------------|
| ----- |    |            |
|       | 5  | 6          |
| ----- |    |            |
|       | 0  | 5          |

Hence,  $46_{10} = 56_8$

So,  $101110_2 = 46_{10} = 56_8$

Thus,  $101110_2 = 56_8$

Example 3.17.  $11010011_2 = ?_{16}$

Solution :

Step 1 : Convert  $11010011_2$  to base 10

Hence,  $211_{10} = D3_{16}$

So,  $11010011_2 = 211_{10} = D3_{16}$

Thus,  $11010011_2 = D3_{16}$

Example 3.16 illustrates the method of converting a number from binary to octal. Similarly, Example 3.17 shows how to convert a number from binary to hexadecimal. However, these are lengthy procedures and shortcut methods can be used when we desire such conversions. We will now discuss these shortcut methods.

### SHORTCUT METHOD FOR BINARY TO OCTAL CONVERSION

The following steps are used in this method :

Step 1 : Divide the binary digits into groups of three (starting from the right).

Step 2 : Convert each group of three binary digits into one octal digit. (Refer to Table 3.1 and try to remember that since there are only 8 digits (0 to 7) in the octal number system, so 3 bits ( $2^3 = 8$ ) are sufficient to represent any octal number in binary). Since decimal digits 0 to 7 are equal to octal digits 0 to 7 so binary to decimal conversion can be used in this step.



Example 3.18.  $101110_2 = ?_8$

Solution :

Step 1 : Divide the binary digits into groups of 3 starting from right (LSD)

101 110

Step 2 : Convert each group into one digit of octal (use binary-to-decimal conversion)

$$\begin{aligned} 101_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 4 + 0 + 1 \\ &= 5_8 \end{aligned}$$

$$\begin{aligned} 110_2 &= 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 \\ &= 4 + 2 + 0 \\ &= 6_8 \end{aligned}$$

Hence,  $101110_2 = 56_8$

Compare this result with the result of Example 3.16.

Example 3.19.  $1101010_2 = ?_8$

Solution :

$$\begin{aligned} 1101010_2 &= \underline{001} \underline{101} \underline{010} \\ &\text{(Group 3 digits from right)} \\ &= 152_8 \\ &\text{(Convert each group to an octal digit)} \end{aligned}$$

Hence,  $1101010_2 = 152_8$

### SHORTCUT METHOD FOR OCTAL TO BINARY CONVERSION

The following steps are used in this method :

Step 1 : Convert each octal digit to a 3 digit binary number. (The octal digits may be treated as decimal for this conversion).

Step 2 : Combine all the resulting binary groups (of 3 digits each) into a single binary number.

Example 3.20.  $562_8 = ?_2$

Solution :

Step 1 : Convert each octal digit to 3 binary digits

$$5_8 = 101_2$$

$$6_8 = 110_2$$

$$2_8 = 010_2$$

Step 2 : Combine the binary groups

$$562_8 = \underline{101} \underline{110} \underline{010}$$

$$5 \quad 6 \quad 2$$

Hence,  $562_8 = 101110010_2$

Example 3.21.  $6751_8 = ?_2$

Solution :

$$6751_8 = \underline{110} \underline{111} \underline{101} \underline{001}$$

$$6 \quad 7 \quad 5 \quad 1$$

$$= 110111101001_2$$

Hence,  $6751_8 = 110111101001_2$

### SHORTCUT METHOD FOR BINARY TO HEXADECIMAL CONVERSION

The following steps are used in this method :

Step 1 : Divide the binary digits into groups of four (starting from the right).

Step 2 : Convert each group of four binary digits to one hexadecimal digit. Remember that hexadecimal digits 0 to 9 are equal to decimal digits 0 to 9 and hexadecimal digits A to F are equal to decimal digits 10 to 15. Hence for this step, the binary-to-decimal conversion procedure can be used, but the decimal values 10 to 15 must be represented as hexadecimal digits A to F.



Example 3.22.  $11010011_2 = ?_{16}$

Solution :

Step 1 : Divide the binary digits into groups of 4.

1101 0011

Step 2 : Convert each group of 4 binary digits to 1 hexadecimal digit.

$$\begin{aligned} 1101_2 &= 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 8 + 4 + 0 + 1 \\ &= 13_{10} \\ &= D_{16} \end{aligned}$$

$$\begin{aligned} 0011_2 &= 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\ &= 0 + 0 + 2 + 1 \\ &= 3_{16} \end{aligned}$$

Hence,  $11010011_2 = D3_{16}$

Compare the result with the result of Example 3.17.

Example 3.23.  $10110101100_2 = ?_{16}$

Solution :

$$10110101100_2 = \underline{0101} \ \underline{1010} \ \underline{1100}$$

(Group 4 digits from right)

$$\begin{aligned} &= 5 \ A \ C \\ &\text{(Convert each group to a hexadecimal digit)} \end{aligned}$$

Hence,  $10110101100_2 = 5AC_{16}$

### SHORTCUT METHOD FOR HEXADECIMAL TO BINARY CONVERSION

The following steps are used in this method :

Step 1 : Convert the decimal equivalent of each hexadecimal digit to 4 binary digits.

Step 2 : Combine all the resulting binary groups (of 4 digits each) into a single binary number.

Example 3.24.  $2AB_{16} = ?_2$

Solution :

Step 1 : Convert the decimal equivalent of each hexadecimal digit to 4 binary digits

$$\begin{aligned} 2_{16} &= 2_{10} = 0010_2 \\ A_{16} &= 10_{10} = 1010_2 \\ B_{16} &= 11_{10} = 1011_2 \end{aligned}$$

Step 2 : Combine the binary groups

$$2AB_{16} = \underline{0010} \ \underline{1010} \ \underline{1011}$$

2    A    B

Hence,  $2AB_{16} = 001010101011_2$

Example 3.25.  $ABC_{16} = ?_2$

Solution :

$$\begin{aligned} ABC_{16} &= \underline{1010} \ \underline{1011} \ \underline{1100} \\ &\quad \quad \quad A \quad B \quad C \\ &= 101010111100_2 \end{aligned}$$

Hence,  $ABC_{16} = 101010111100_2$

Finally, Table 3.2 summarises the relationship between the decimal, binary, hexadecimal, and octal number systems. Note that the maximum value for a single digit of octal (7) is equal to the maximum value of three digits of binary. The value range of one digit of octal duplicates the value range of three digits of binary. If octal digits are substituted for binary digits, the substitution is on a one-to-three basis. Thus, computers that print octal numbers instead of binary, while taking memory dump, save one-third of the printing space and time.

Similarly, note that the maximum value of one digit in hexadecimal is equal to the maximum value of four digits in binary. Thus the value range of one digit of hexadecimal is equivalent to the value range of four digits of binary. Therefore, hexadecimal shortcut notation is a one-to-four reduction in the space and time required for memory dump.



FRACTIONAL NUMBERS

In binary number system, fractional numbers are formed in the same general way as in the decimal system. Just as in the decimal system,

0.235 = (2x10<sup>-1</sup>) + (3x10<sup>-2</sup>) + (5x10<sup>-3</sup>)  
and  
68.53 = (6x10<sup>1</sup>) + (8x10<sup>0</sup>) + (5x10<sup>-1</sup>) + (3x10<sup>-2</sup>)

Similarly in the binary system,

0.101 = (1x2<sup>-1</sup>) + (0x2<sup>-2</sup>) + (1x2<sup>-3</sup>)  
and  
10.01 = (1x2<sup>1</sup>) + (0x2<sup>0</sup>) + (0x2<sup>-1</sup>) + (1x2<sup>-2</sup>)

Thus, the binary point serves the same purpose as the decimal point. Some of the positional values in the binary system are given below.

|                      |                   |                |                |                |                |   |                 |                 |                 |                 |
|----------------------|-------------------|----------------|----------------|----------------|----------------|---|-----------------|-----------------|-----------------|-----------------|
|                      | Binary Point<br>↓ |                |                |                |                |   |                 |                 |                 |                 |
| Position             | 4                 | 3              | 2              | 1              | 0              | . | -1              | -2              | -3              | -4              |
| Position Value       | 2 <sup>4</sup>    | 2 <sup>3</sup> | 2 <sup>2</sup> | 2 <sup>1</sup> | 2 <sup>0</sup> |   | 2 <sup>-1</sup> | 2 <sup>-2</sup> | 2 <sup>-3</sup> | 2 <sup>-4</sup> |
| Quantity Represented | 16                | 8              | 4              | 2              | 1              |   | $\frac{1}{2}$   | $\frac{1}{4}$   | $\frac{1}{8}$   | $\frac{1}{16}$  |

In general, a number in a number system with base b would be written as :

$a_n a_{n-1} \dots a_0 . a_{-1} a_{-2} \dots a_{-m}$

and would be interpreted to mean

$a_n \times b^n + a_{n-1} \times b^{n-1} + \dots + a_0 \times b^0 + a_{-1} \times b^{-1} + a_{-2} \times b^{-2} + \dots + a_{-m} \times b^{-m}$

The symbols  $a_n, a_{n-1}, \dots, a_{-m}$  used in the above representation should be one of the b symbols allowed in the number system.

Thus, as per the above mentioned general rule,

$46.32_8 = (4 \times 8^1) + (6 \times 8^0) + (3 \times 8^{-1}) + (2 \times 8^{-2})$

and

$5A.3C_{16} = (5 \times 16^1) + (A \times 16^0) + (3 \times 16^{-1}) + (C \times 16^{-2})$

Example 3.26. Find the decimal equivalent of the binary number 110.101

Solution :

$110.101_2 = 1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$   
 $= 4 + 2 + 0 + .5 + 0 + .125$   
 $= 6 + 0.5 + 0.125$   
 $= 6.625_{10}$

Example 3.27. Find the decimal equivalent of the octal number 127.54

Solution :

$127.54_8 = 1 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 + 5 \times 8^{-1} + 4 \times 8^{-2}$   
 $= 64 + 16 + 7 + 5/8 + 4/64$   
 $= 87 + 0.625 + 0.0625$   
 $= 87.6875_{10}$

Example 3.28. Find the decimal equivalent of the hexadecimal number 2B.C4

Solution :

$2B.C4_{16} = 2 \times 16^1 + B \times 16^0 + C \times 16^{-1} + 4 \times 16^{-2}$   
 $= 32 + B + C/16 + 4/256$   
 $= 43 + 0.75 + 0.015625$   
 $= 43.765625_{10}$



Table 3.2. Relationship between Decimal, Binary, Hexadecimal and Octal Number Systems

| Decimal | Hexa<br>Decimal | Binary | Octal |
|---------|-----------------|--------|-------|
| 0       | 0               | 0      | 0     |
| 1       | 1               | 1      | 1     |
| 2       | 2               | 10     | 2     |
| 3       | 3               | 11     | 3     |
| 4       | 4               | 100    | 4     |
| 5       | 5               | 101    | 5     |
| 6       | 6               | 110    | 6     |
| 7       | 7               | 111    | 7     |
| 8       | 8               | 1000   | 10    |
| 9       | 9               | 1001   |       |
| 10      | A               | 1010   |       |
| 11      | B               | 1011   |       |
| 12      | C               | 1100   |       |
| 13      | D               | 1101   |       |
| 14      | E               | 1110   |       |
| 15      | F               | 1111   |       |
| 16      | 10              | 10000  |       |

## QUESTIONS

- What is the difference between a positional and a non-positional number system? Give examples of both types of number systems.
- What is meant by the base of a number system? Give examples to illustrate the role of base in positional number systems.
- What is the value of the base for decimal, hexadecimal, binary and octal number systems?
- Give an example for octal number system to show that the same digit may signify different values depending on the position it occupies in the number.
- What will be the total number of different symbols or digits and the maximum value of a single digit for the following number systems :
  - number system with base 5.
  - number system with base 20.
  - number system with base 9.
  - number system with base 12.
- What is a bit in computer terminology? How many different patterns of bits are possible with-
  - 6 bits
  - 7 bits
  - 8 bits?
- Explain the meaning of the term "memory dump".
- Why are octal and/or hexadecimal number systems used as shortcut notation?
- Find out the decimal equivalents of the following binary numbers.
 

|                |             |
|----------------|-------------|
| a) 1101011     | b) 11010    |
| c) 10110011    | d) 11011101 |
| e) 1110101     | f) 1000     |
| g) 10110001100 |             |
| h) 1010101100  |             |
| i) 110001      | j) 111      |
- Find out the octal equivalents of the binary numbers of Question 9.
- Find out the hexadecimal equivalents of the binary numbers of Question 9.



12. Convert the following numbers to decimal :

- a)  $110110_2$                       b)  $2573_6$
- c)  $2A3B_{16}$                       d)  $1234_9$

13. Convert the following decimal numbers to binary :

- a)  $435_{10}$                       b)  $1694_{10}$
- c)  $32_{10}$                       d)  $135_{10}$

14. Convert the decimal numbers of Question 13 to octal.

15. Convert the decimal numbers of Question 13 to hexadecimal.

16. a)  $125_6 = ?_4$   
b)  $24_9 = ?_3$   
c)  $ABC_{16} = ?_8$

17. Convert the following numbers to their binary equivalent :

- a)  $2AC_{16}$                       b)  $FAB_{16}$
- c)  $2614_8$                       d)  $562_8$

18. Find the decimal equivalent of the following numbers

- a)  $111.01_2$                       b)  $1001.011_2$
- c)  $247.65_8$                       d)  $A2B.D4_{16}$



## CHAPTER 4

## 4. COMPUTER CODES

In the previous chapter, we have discussed about true or "pure" binary numbers. In this chapter, we will see how these binary numbers are coded to represent characters in the computer memory. Thus, the goal of this chapter is to present the formats used in computer memory to record data. Although many coding schemes have been developed over the years, we will be discussing only the most commonly used computer codes.

Numeric data is not the only form of data that is to be handled by a computer. We often require to process alphanumeric data also. An *alphanumeric data* is a string of symbols where a symbol may be one of the letters A,B,C,...,Z or one of the digits 0,1,2,...,9 or a *special character*, such as + - \* / , . ( ) = (space or blank) etc. An *alphabetic data* consists of only the letters A,B,C,...,Z and the blank character. Similarly, *numeric data* consists of only numbers 0,1,2,...,9. However, any data must be represented internally by the bits 0 & 1. As such, binary coding schemes are used to represent data internally in the computer memory. In binary coding, every symbol that appears in the data is represented by a group of bits. The group of bits used to represent a symbol is called a byte. To indicate the number of bits in a group, sometimes a byte is

referred to as "n-bit byte" where the group contains n bits. However, the term byte is commonly used to mean an 8-bit byte (a group of 8 bits) because most of the modern computers use 8 bits to represent a symbol.

## BCD CODE

The Binary Coded Decimal (BCD) code is one of the early memory codes. It is based on the idea of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. This facilitates the conversion process to a great extent.

The BCD equivalent of each decimal digit is shown in Table 4.1. Since 8 and 9 require 4 bits, all decimal digits are represented in BCD by 4 bits. You have seen in Example 3.9 that  $42_{10}$  is equal to  $101010_2$  in a pure binary form.

Converting  $42_{10}$  into BCD, however, produces the following result :

$$42_{10} = \begin{array}{cc} 0100 & 0010 \\ 4 & 2 \end{array}$$

or 01000010 in BCD



Note that each decimal digit is independently converted to a 4 bit binary number and hence the conversion process is very easy. Also note that when 4 bits are used, altogether 16 ( $2^4$ ) configurations are possible (refer to hexadecimal number system). But from Table 4.1 you can see that only the first 10 of these combinations are used to represent decimal digits. The remaining 6 arrangements (1010,

Table 4.1. BCD Equivalent of Decimal Digits

| Decimal Digits | BCD Equivalent |
|----------------|----------------|
| 0              | 0000           |
| 1              | 0001           |
| 2              | 0010           |
| 3              | 0011           |
| 4              | 0100           |
| 5              | 0101           |
| 6              | 0110           |
| 7              | 0111           |
| 8              | 1000           |
| 9              | 1001           |

1011, 1100, 1101, 1110 and 1111) have decimal values from 10 to 15. These arrangements are not used in BCD coding. That is, 1010 does not represent  $10_{10}$  in BCD. Instead,

$$10_{10} = \begin{array}{cc} \underline{0001} & \underline{0000} \\ 1 & 0 \end{array}$$

or 00010000 in BCD

Similarly,

$$15_{10} = \begin{array}{cc} \underline{0001} & \underline{0101} \\ 1 & 5 \end{array}$$

or 00010101 in BCD

In the above discussion, we have used a group of 4 bits to represent a digit (character) in BCD. 4-bit BCD coding system can be used to represent only decimal numbers because 4 bits are insufficient to represent the various characters used by a computer. Instead of using 4 bits with only 16 possible characters, computer designers commonly use 6 bits to represent characters in BCD code. In the 6-bit BCD code, the four BCD numeric place positions are retained, but two additional *zone* positions are added. With 6 bits, it is possible to represent 64 ( $2^6$ )

different characters. This is a sufficient number to code decimal digits (10), alphabetic letters (26), and other special characters (28). Table 4.2 illustrates the coding of alphabetic and numeric characters in BCD.

In Chapter 3, we have seen the use of octal and hexadecimal number systems as shortcut notation for binary. Because BCD is a 6-bit code, it can be easily divided into two 3-bit groups. Each of these 3-bit groups can be represented by 1 octal digit. Thus, octal number system is used as shortcut notation for memory dump in computers that use BCD code for internal representation of characters. This results in a one-to-three reduction in the volume of memory dump. Table 4.2 also shows the octal equivalent of the alphabetic and numeric characters coded in BCD.

**Example 4.1.** Show the binary digits used to record the word BASE in BCD.

**Solution :**

B = 110010 in BCD binary notation  
A = 110001 in BCD binary notation  
S = 010010 in BCD binary notation  
E = 110101 in BCD binary notation

So the binary digits

$\begin{array}{cccc} \underline{110010} & \underline{110001} & \underline{010010} & \underline{110101} \\ B & A & S & E \end{array}$

will record the word BASE in BCD.

**Example 4.2.** Using octal notation show the BCD coding for the word DIGIT.

**Solution :**

D = 64 in BCD octal notation  
I = 71 in BCD octal notation  
G = 67 in BCD octal notation  
I = 71 in BCD octal notation  
T = 23 in BCD octal notation

So the BCD coding for the word DIGIT in octal notation will be :

$\begin{array}{ccccc} \underline{64} & \underline{71} & \underline{67} & \underline{71} & \underline{23} \\ D & I & G & I & T \end{array}$



Table 4.2. **Alphabetic And Numeric Characters In BCD Along With Their Octal Equivalent.**

| Character | BCD Code |       | Octal Equivalent |
|-----------|----------|-------|------------------|
|           | Zone     | Digit |                  |
| A         | 11       | 0001  | 61               |
| B         | 11       | 0010  | 62               |
| C         | 11       | 0011  | 63               |
| D         | 11       | 0100  | 64               |
| E         | 11       | 0101  | 65               |
| F         | 11       | 0110  | 66               |
| G         | 11       | 0111  | 67               |
| H         | 11       | 1000  | 70               |
| I         | 11       | 1001  | 71               |
| J         | 10       | 0001  | 41               |
| K         | 10       | 0010  | 42               |
| L         | 10       | 0011  | 43               |
| M         | 10       | 0100  | 44               |
| N         | 10       | 0101  | 45               |
| O         | 10       | 0110  | 46               |
| P         | 10       | 0111  | 47               |
| Q         | 10       | 1000  | 50               |
| R         | 10       | 1001  | 51               |
| S         | 01       | 0010  | 22               |
| T         | 01       | 0011  | 23               |
| U         | 01       | 0100  | 24               |
| V         | 01       | 0101  | 25               |
| W         | 01       | 0110  | 26               |
| X         | 01       | 0111  | 27               |
| Y         | 01       | 1000  | 30               |
| Z         | 01       | 1001  | 31               |
| 1         | 00       | 0001  | 01               |
| 2         | 00       | 0010  | 02               |
| 3         | 00       | 0011  | 03               |
| 4         | 00       | 0100  | 04               |
| 5         | 00       | 0101  | 05               |
| 6         | 00       | 0110  | 06               |
| 7         | 00       | 0111  | 07               |
| 8         | 00       | 1000  | 10               |
| 9         | 00       | 1001  | 11               |
| 0         | 00       | 1010  | 12               |

## EBCDIC

The major problem with BCD code is that only 64 ( $2^6$ ) different characters can be represented in it. This is not sufficient for providing decimal numbers (10), lower-case letters (26), capital letters (26), and a fairly large number of other special characters (28+).

Hence, the BCD code was extended from a 6-bit code to an 8-bit code. The added 2 bits are used as additional zone bits, expanding the zone to 4 bits. The resulting code is called the *extended binary-coded decimal interchange code (EBCDIC)*. In this code, it is possible to represent 256 ( $2^8$ ) different characters instead of 64 ( $2^6$ ). In addition to the various character requirements mentioned above, this also allows a large variety of printable characters and several nonprintable control characters. The *control characters* are used to control such activities as printer vertical spacing, movement of cursor on the terminal screen, etc. All of the 256 bit combinations have not yet been assigned characters, so the code can still grow as new requirements develop.

Because EBCDIC is an 8-bit code, it can be easily divided into two 4-bit groups. Each of these 4-bit groups can be represented by 1 hexadecimal digit (refer to Chapter 3). Thus, hexadecimal number system is used as shortcut notation for memory dump by computers that use EBCDIC for internal representation of characters. This results in a one-to-four reduction in the volume of memory dump. Table 4.3 shows the alphabetic and numeric characters in EBCDIC along with their hexadecimal equivalent.

Developed by IBM, EBCDIC code is used in most IBM models and in many other computers.

## ZONED AND PACKED DECIMAL NUMBERS

From Table 4.3, it can be observed that in the EBCDIC code, the digit values are the same as the numeric characters - 0 through 9 (0000-1001). But numeric values need some special consideration because we must have a way of indicating whether the number is positive, negative, or unsigned (implies positive). Thus, when a numeric value is represented in EBCDIC, a sign indicator is used in the zone position of the rightmost digit. A sign indicator of hexadecimal C is a plus sign, hexadecimal D is a minus sign, and a hexadecimal F means the number is unsigned. Table 4.4 illustrates the representation of numeric values in EBCDIC. Note that the only zone affected by the sign is the zone of the rightmost digit. All other zones remain as F, the zone value for numeric characters in EBCDIC. Because each decimal digit has a zone with it, numbers coded in EBCDIC are called *zoned decimal numbers*. Numeric data



input into the computer are usually zoned decimal numbers. Printers can print only those numeric characters that are in a zoned-decimal format.

However, most computers cannot perform arithmetic operations on zoned-decimal data. Thus, before any arithmetic operation can be performed, the data must be converted to a format on which arithmetic operations are possible. One such acceptable format is the *packed decimal* format. The following steps are used to convert a zoned decimal number to a packed decimal number :

**Step 1 :** The zone half and the digit half of the rightmost byte are reversed. This moves the sign to the extreme right of the number.

**Step 2 :** All remaining zones are dropped out.

Table 4.5 illustrates the conversion process of zoned decimal data to packed data.

It may be observed that packed data requires fewer number of bytes (group of 8 bits) as compared to zoned data. In the zoned format, there is only one digit per byte (each digit along with the zone requires 8 bits). But there are two digits in each byte in the packed format (each digit requires 4 bits). If the packing process does not completely fill a byte, it is filled with a zero. For example, the zoned data F3F4F5F6 will convert to packed data 03456F. Observe that in this example, the zoned data requires 4 bytes and the packed data requires only 3 bytes.

**Example 4.3.** Using binary notation, write the EBCDIC coding for the word BIT. How many bytes are required for this representation?

**Solution :**

B = 1100 0010 in EBCDIC binary notation  
I = 1100 1001 in EBCDIC binary notation  
T = 1110 0011 in EBCDIC binary notation

So the EBCDIC coding for the word BIT in binary notation will be :

11000010 11001001 11100011  
B I T

3 bytes will be required for this representation because each letter requires 1 byte (or 8 bits) for its representation.

Table 4.3. Alphabetic And Numeric Characters In EBCDIC Along With Their Hexadecimal Equivalent.

| Character | EBCDIC Code |       | Hexadecimal Equivalent |
|-----------|-------------|-------|------------------------|
|           | Zone        | Digit |                        |
| A         | 1100        | 0001  | C1                     |
| B         | 1100        | 0010  | C2                     |
| C         | 1100        | 0011  | C3                     |
| D         | 1100        | 0100  | C4                     |
| E         | 1100        | 0101  | C5                     |
| F         | 1100        | 0110  | C6                     |
| G         | 1100        | 0111  | C7                     |
| H         | 1100        | 1000  | C8                     |
| I         | 1100        | 1001  | C9                     |
| J         | 1101        | 0001  | D1                     |
| K         | 1101        | 0010  | D2                     |
| L         | 1101        | 0011  | D3                     |
| M         | 1101        | 0100  | D4                     |
| N         | 1101        | 0101  | D5                     |
| O         | 1101        | 0110  | D6                     |
| P         | 1101        | 0111  | D7                     |
| Q         | 1101        | 1000  | D8                     |
| R         | 1101        | 1001  | D9                     |
| S         | 1110        | 0010  | E2                     |
| T         | 1110        | 0011  | E3                     |
| U         | 1110        | 0100  | E4                     |
| V         | 1110        | 0101  | E5                     |
| W         | 1110        | 0110  | E6                     |
| X         | 1110        | 0111  | E7                     |
| Y         | 1110        | 1000  | E8                     |
| Z         | 1110        | 1001  | E9                     |
| 0         | 1111        | 0000  | F0                     |
| 1         | 1111        | 0001  | F1                     |
| 2         | 1111        | 0010  | F2                     |
| 3         | 1111        | 0011  | F3                     |
| 4         | 1111        | 0100  | F4                     |
| 5         | 1111        | 0101  | F5                     |
| 6         | 1111        | 0110  | F6                     |
| 7         | 1111        | 0111  | F7                     |
| 8         | 1111        | 1000  | F8                     |
| 9         | 1111        | 1001  | F9                     |



Table 4.4. Numeric Values In EBCDIC In Hexadecimal Notation

| Numeric Value | EBCDIC | Sign Indicator |
|---------------|--------|----------------|
| 345           | F3F4F5 | F for unsigned |
| +345          | F3F4C5 | C for positive |
| -345          | F3F4D5 | D for negative |

Table 4.5. Zoned And Packed Decimal Numbers

| Numeric Value | Zoned Format | Packed Format |
|---------------|--------------|---------------|
| 345           | F3F4F5       | 345F          |
| +345          | F3F4C5       | 345C          |
| -345          | F3F4D5       | 345D          |
| 3456          | F3F4F5F6     | 03456F        |

**Example 4.4.** Write the EBCDIC coding for the word ZONE (use hexadecimal notation). How many bytes will be required for this representation ?

**Solution :**

Z = E9 in EBCDIC hexadecimal notation  
 O = D6 in EBCDIC hexadecimal notation  
 N = D5 in EBCDIC hexadecimal notation  
 E = C5 in EBCDIC hexadecimal notation.

So the EBCDIC coding for the word ZONE in hexadecimal notation will be :

E9 D6 D5 C5  
 Z O N E

Each hexadecimal digit requires 4 bits and there are altogether 8 hexadecimal digits. So in all  $8 \times 4 = 32$  bits will be required. But 8 bits = 1 byte. So 32 bits = 4 bytes. Hence, 4 bytes will be required for this representation.

We may also write directly that since each letter requires 1 byte for its representation in EBCDIC and there are 4 letters in the word ZONE, so 4 bytes will be required.

for this representation.

**Example 4.5.** Write the EBCDIC zoned-decimal coding for the value +256 (use hexadecimal). How many bytes will be required for this representation ?

**Solution :**

+256 = F2F5C6 in EBCDIC

Each hexadecimal digit requires 4 bits and there are altogether 6 hexadecimal digits. So in all  $6 \times 4 = 24$  bits or 3 bytes (8 bits = 1 byte) will be required for this representation.

We may also write directly that since each digit requires 1 byte for its representation in the EBCDIC zoned decimal coding and there are 3 digits in the given number, so 3 bytes will be required for this representation.

**Example 4.6.** Write -128 as packed decimal number (use hexadecimal). How many bytes will be required for this representation?

**Solution :**

-128 = F1F2D8 in EBCDIC  
 = 128D in packed format.

Each hexadecimal digit requires 4 bits and there are altogether 4 hexadecimal digits. So in all  $4 \times 4 = 16$  bits or 2 bytes (1 byte = 8 bits) will be required for this representation.

## ASCII

Another computer code that is very widely used is the American Standard Code for Information Interchange (ASCII). ASCII has been adopted by several American computer manufacturers as their computers' internal code. This code is popular in data communications, is used almost exclusively to represent data internally in microcomputers, and is frequently found in the larger computers produced by some vendors.

ASCII is of two types : ASCII-7 and ASCII-8. ASCII-7 is a 7 bit code that allows 128 ( $2^7$ ) different characters. The first 3 bits are used as zone bits and the last 4 bits indicate the digit. Microcomputers using 8-bit byte (group of 8 bits for one byte) use the 7-bit ASCII by leaving the leftmost first bit of each byte as a zero. Table



4.6 shows the alphabetic and numeric characters in ASCII-7 notation.

ASCII-8 is an extended version of ASCII-7. It is an 8-bit code that allows 256 ( $2^8$ ) different characters rather than 128. The additional bit is added to the zone bits. Table 4.7 shows the alphabetic and numeric characters in ASCII-8 notation. Observe that other than the zone-value differences, ASCII-7 and ASCII-8 are identical. ASCII also uses hexadecimal as its four-to-one shortcut notation for memory dump. Tables 4.6 and 4.7 also show the hexadecimal equivalent of the ASCII notations.

**Example 4.7.** Write the binary coding for the word BOY in ASCII-7. How many bytes are required for this representation?

**Solution :**

B = 1000010 in ASCII-7 binary notation

O = 1001111 in ASCII-7 binary notation

Y = 1011001 in ASCII-7 binary notation.

Hence the binary coding for the word BOY in ASCII-7 will be :

|                |                |                |
|----------------|----------------|----------------|
| <u>1000010</u> | <u>1001111</u> | <u>1011001</u> |
| B              | O              | Y              |

As each character in ASCII-7 requires one byte for its representation and since there are 3 characters in the word BOY, so 3 bytes will be required for this representation.

**Example 4.8.** Write the hexadecimal coding for the word GIRL in ASCII-7. How many bytes are required for this representation?

**Solution :**

G = 47 in ASCII-7 hexadecimal notation

I = 49 in ASCII-7 hexadecimal notation

R = 52 in ASCII-7 hexadecimal notation

L = 4C in ASCII-7 hexadecimal notation

Hence the hexadecimal coding for the word GIRL in ASCII-7 will be :

|           |           |           |           |
|-----------|-----------|-----------|-----------|
| <u>47</u> | <u>49</u> | <u>52</u> | <u>4C</u> |
| G         | I         | R         | L         |

As each character in ASCII-7 requires one byte for its representation and since there are 4 characters in the word

Table 4.6. Numeric And Alphabetic Characters In ASCII-7 Notation Along With Their Hexadecimal Equivalent.

| Character | ASCII-7 Code |       | Hexadecimal Equivalent |
|-----------|--------------|-------|------------------------|
|           | Zone         | Digit |                        |
| 0         | 011          | 0000  | 30                     |
| 1         | 011          | 0001  | 31                     |
| 2         | 011          | 0010  | 32                     |
| 3         | 011          | 0011  | 33                     |
| 4         | 011          | 0100  | 34                     |
| 5         | 011          | 0101  | 35                     |
| 6         | 011          | 0110  | 36                     |
| 7         | 011          | 0111  | 37                     |
| 8         | 011          | 1000  | 38                     |
| 9         | 011          | 1001  | 39                     |
|           |              |       |                        |
| A         | 100          | 0001  | 41                     |
| B         | 100          | 0010  | 42                     |
| C         | 100          | 0011  | 43                     |
| D         | 100          | 0100  | 44                     |
| E         | 100          | 0101  | 45                     |
| F         | 100          | 0110  | 46                     |
| G         | 100          | 0111  | 47                     |
| H         | 100          | 1000  | 48                     |
| I         | 100          | 1001  | 49                     |
| J         | 100          | 1010  | 4A                     |
| K         | 100          | 1011  | 4B                     |
| L         | 100          | 1100  | 4C                     |
| M         | 100          | 1101  | 4D                     |
| N         | 100          | 1110  | 4E                     |
| O         | 100          | 1111  | 4F                     |
|           |              |       |                        |
| P         | 101          | 0000  | 50                     |
| Q         | 101          | 0001  | 51                     |
| R         | 101          | 0010  | 52                     |
| S         | 101          | 0011  | 53                     |
| T         | 101          | 0100  | 54                     |
| U         | 101          | 0101  | 55                     |
| V         | 101          | 0110  | 56                     |
| W         | 101          | 0111  | 57                     |
| X         | 101          | 1000  | 58                     |
| Y         | 101          | 1001  | 59                     |
| Z         | 101          | 1010  | 5A                     |



representation.

**Example 4.9.** Write the binary coding for the word SKY in ASCII-8. How many bytes are required for this representation?

**Solution :**

S = 10110011 in ASCII-8 binary notation  
K = 10101011 in ASCII-8 binary notation  
Y = 10111001 in ASCII-8 binary notation

Hence the binary coding for the word SKY in ASCII-8 will be :

10110011 10101011 10111001  
S K Y

As each character in ASCII-8 requires one byte for its representation and since there are 3 characters in the word SKY, so 3 bytes will be required for this representation.

**Example 4.10.** Write the hexadecimal coding for the word STAR in ASCII-8. How many bytes are required for this representation?

**Solution :**

S = B3 in ASCII-8 hexadecimal notation  
T = B4 in ASCII-8 hexadecimal notation  
A = A1 in ASCII-8 hexadecimal notation  
R = B2 in ASCII-8 hexadecimal notation

Hence the hexadecimal coding for the word STAR in ASCII-8 will be :

B3 B4 A1 B2  
S T A R

As each character in ASCII-8 requires one byte for its representation and since there are 4 characters in the word STAR, so 4 bytes will be required for this representation.

## COLLATING SEQUENCE

The value of an alphanumeric or alphabetic data element is usually the name of some object. Obviously one would not like to perform any arithmetic on such data but one may like to compare them in order to arrange them in some desired sequence. Now, if we compare the alphabetic

values A and B, which one will be treated as greater by the computer? For an answer to such questions, it is necessary to have some assigned ordering among the characters used by the computer. This ordering is known as the *collating sequence*.

Collating sequence may vary from one computer system to another depending on the type of computer code used by a particular computer. To illustrate this, let us consider the computer codes already discussed in this chapter. Observe from Tables 4.2 and 4.3 that the zone values of the characters A through 9 decreases in BCD code from the equivalent of decimal 3 down to 0, while in EBCDIC, the zone values of the characters A through 9 increases from the equivalent of decimal 12 to 15. This means that a computer which uses BCD code for its internal representation of characters will treat alphabetic characters (A, B, ..., Z) to be greater than numeric characters (0, 1, ..., 9). On the other hand, a computer which uses EBCDIC for its internal representation of characters will treat numeric characters to be greater than alphabetic characters. Similarly, observe from Tables 4.6 and 4.7 that a computer which uses ASCII for its internal representation of characters will place numbers ahead of letters during a sort (ascending) because the number characters have a zone value that is less than the zone value for letters.

However, whatever may be the type of computer code used, in most (not all - in BCD 0>9) collating sequences the following rules are observed :

1. Letters are considered in alphabetic order (A<B<C<...<Z)
2. Digits are considered in numeric order (0<1<2<...<9).

**Example 4.11.** Suppose a computer uses EBCDIC as its internal representation of characters. In which order will this computer sort the following strings : 23, A1, 1A ?

**Solution :**

In EBCDIC, numeric characters are treated to be greater than alphabetic characters. Hence the numeric characters will be placed after the alphabetic characters. So the computer will treat the given string as :

A1<1A<23

Hence the sorted sequence will be : A1, 1A, 23..



**Table 4.7. Numeric And Alphabetic Characters In ASCII-8 Notation Along With Their Hexadecimal Equivalent.**

| Character | ASCII-8 Code |       | Hexadecimal Equivalent |
|-----------|--------------|-------|------------------------|
|           | Zone         | Digit |                        |
| 0         | 0101         | 0000  | 50                     |
| 1         | 0101         | 0001  | 51                     |
| 2         | 0101         | 0010  | 52                     |
| 3         | 0101         | 0011  | 53                     |
| 4         | 0101         | 0100  | 54                     |
| 5         | 0101         | 0101  | 55                     |
| 6         | 0101         | 0110  | 56                     |
| 7         | 0101         | 0111  | 57                     |
| 8         | 0101         | 1000  | 58                     |
| 9         | 0101         | 1001  | 59                     |
|           |              |       |                        |
| A         | 1010         | 0001  | A1                     |
| B         | 1010         | 0010  | A2                     |
| C         | 1010         | 0011  | A3                     |
| D         | 1010         | 0100  | A4                     |
| E         | 1010         | 0101  | A5                     |
| F         | 1010         | 0110  | A6                     |
| G         | 1010         | 0111  | A7                     |
| H         | 1010         | 1000  | A8                     |
| I         | 1010         | 1001  | A9                     |
| J         | 1010         | 1010  | AA                     |
| K         | 1010         | 1011  | AB                     |
| L         | 1010         | 1100  | AC                     |
| M         | 1010         | 1101  | AD                     |
| N         | 1010         | 1110  | AE                     |
| O         | 1010         | 1111  | AF                     |
|           |              |       |                        |
| P         | 1011         | 0000  | B0                     |
| Q         | 1011         | 0001  | B1                     |
| R         | 1011         | 0010  | B2                     |
| S         | 1011         | 0011  | B3                     |
| T         | 1011         | 0100  | B4                     |
| U         | 1011         | 0101  | B5                     |
| V         | 1011         | 0110  | B6                     |
| W         | 1011         | 0111  | B7                     |
| X         | 1011         | 1000  | B8                     |
| Y         | 1011         | 1001  | B9                     |
| Z         | 1011         | 1010  | BA                     |

**Example 4.12.**

Suppose a computer uses ASCII for its internal representation of characters. In which order will the computer sort the following strings 23, A1, 1A ?

**Solution :**

In ASCII, numeric characters are treated to be less than alphabetic characters. Hence the numeric character will be placed before the alphabetic characters. So the computer will treat the given string as : 23 < 1A < A1.

Hence the sorted sequence will be : 23, 1A, A1.

## QUESTIONS

1. Define the term 'byte'. What is the difference between a bit and a byte ?
2. Write the 4-bit BCD code for the following numbers : (a)  $25_{10}$  (b)  $64_{10}$  (c)  $128_{10}$  (d)  $1024_{10}$
3. Using binary notation, show the BCD coding for the following words : (a) BIT (b) BYTE (c) CODE.
4. Using octal notation, show the BCD coding for the following words : (a) COMPUTER INPUT (c) VIDEO.
5. Why was BCD code extended to EBCDIC ?
6. How many different characters are possible for the following codes : BCD, EBCDIC, ASCII and ASCII-8 ?
7. Suppose a new computer code is designed that uses 9 bits. How many different characters are possible in this code ?
8. Why are octal and hexadecimal shorthand notations used ? Identify the shorthand notation used for each of these computer codes : BCD, EBCDIC, ASCII-7, and ASCII-8.
9. Why do we have a packed decimal format ? How does it differ from a zoned decimal format ?



10. Using binary notation, write the EBCDIC coding for the following words : (a) SUN (b) MOON (c) AT. How many bytes are required for each of these representations ?
11. Using hexadecimal notation, write the EBCDIC coding for the following words : (a) PROGRAM (b) OUTPUT (c) BYTE. How many bytes are required for each of these representations ?
12. Using hexadecimal notation, write the zoned-decimal coding for the following numbers : (a) 1256 (b) +439 (c) -63. How many bytes will be required for each of these representations ?
13. Using hexadecimal notation, write the packed-decimal coding for the following numbers : (a) 12915 (b) +9876 (c) 872.  
How many bytes will be required for each of these representations ?
14. List out the similarities and differences between 7-bit and 8-bit ASCII.
15. Using binary notation, write the ASCII-7 and ASCII-8 codes for the following words : (a) DRY (b) WET (c) DAMP.
- How many bytes are required for each of these representations ?
16. Using hexadecimal notation, write the ASCII-7 and ASCII-8 codes for the following words : (a) PRINT (b) TYPE (c) RUB.  
How many bytes are required for each of these representations ?
17. Explain the meaning of the term "collating sequence".
18. A computer uses EBCDIC as its internal representation of characters. In which order will this computer sort the following strings : ABC, 123, 245, ADD ?
19. A computer uses ASCII. In which order will this computer sort the following strings : BED, 512, ADD, 128, BAD ?
20. Give the full form of the following abbreviations : BCD, EBCDIC, ASCII.







## CHAPTER 5

# 5. COMPUTER ARITHMETIC

In Chapter 4 you have seen that computers store numbers, letters, and other characters in coded form that is related to the binary number system. In this chapter you will learn why computers use binary numbers instead of decimal numbers and how the basic arithmetic operations are performed inside the computer using binary numbers.

### WHY BINARY

You might have observed in Chapter 3 that the use of a smaller base may require more positions to represent a given value (recall the reason for using octal and hexadecimal shortcut notations). For example,  $9_{10} = 1001_2$ . Here four positions are required instead of one to represent the decimal number 9 in binary form. In spite of this fact, almost all computers use binary numbers. So the obvious question that arises to one's mind is that why do we go for binary numbers instead of decimal numbers? The reasons are as follows :

1. The first and the foremost reason is that electronic and electrical components, by their very nature, operate in a binary mode. Information is handled in the computer by

electronic/electrical components such as transistors, semiconductors, wires, etc. all of which can only indicate two states or conditions - on(1) or off(0). Transistors are either conducting(1) or nonconducting(0) ; magnetic materials are either magnetized(1) or non-magnetized(0) in one direction or in the opposite direction; a pulse or voltage is present(1) or not present(0) in wire. All information is represented within the computer by the presence or absence of these various signals. The binary number system, which has only two digits (0 and 1), is most suitable and is conveniently used to express the two possible states. The concept of binary components is illustrated in Figure 5.1.

2. The second reason is that computer circuits only have to handle two binary digits rather than ten decimal digits. The result is that the internal circuit design of computers is simplified to a great extent. This ultimately results in less expensive and more reliable circuits for computers.



3. Finally, the binary system is used because everything that can be done with a base of 10 can also be done in binary. How this is achieved has been discussed below.

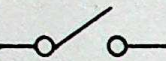
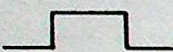
| BINARY STATE  | ON (1)                                                                            | OFF (0)                                                                           |
|---------------|-----------------------------------------------------------------------------------|-----------------------------------------------------------------------------------|
| Bulb          |  |  |
| Switch        |  |  |
| Circuit Pulse |  |  |

Figure 5.1. Examples of devices that work in binary mode. These devices can only represent two states - on or off. These states can represent either a 1 or a 0 or a yes or a no.

## BINARY ARITHMETIC

In this section you will see how the four basic arithmetic operations are performed inside a computer using binary numbers. Actually, binary arithmetic is much simpler to learn because binary system deals with only two digits - 0 and 1. So all binary numbers are made up of only 0's and 1's and when arithmetic operations are performed on these numbers, the results are also in 0's and 1's only.

### ADDITION

Binary addition is performed in the same manner as decimal addition. However, since binary system has only two digits, the addition table for binary arithmetic is very simple consisting of only four entries. The complete table for binary addition is as follows:

$$\begin{aligned}
 0 + 0 &= 0 \\
 0 + 1 &= 1 \\
 1 + 0 &= 1 \\
 1 + 1 &= 0 \text{ plus a carry of 1 to next higher column}
 \end{aligned}$$

Carry-overs are performed in the same manner as in decimal arithmetic. Since 1 is the largest digit in the binary system, any sum greater than 1 requires that a digit be carried over. For instance, 10 plus 10 binary requires the addition of two 1's in the second position. Since  $1+1=0$  plus a carry-over of 1, the sum of  $10 + 10$  is 100 in binary.

By repeated use of the above rules, any two binary numbers can be added together by adding two bits at a time. The exact procedure is illustrated with the examples given below.

*Example 5.1.*

Add the binary numbers 101 and 10 in both decimal and binary form.

*Solution :*

| BINARY | DECIMAL |
|--------|---------|
| 101    | 5       |
| +10    | +2      |
| ----   | ----    |
| 111    | 7       |

*Example 5.2.*

Add the binary numbers 10011 and 1001 in both decimal and binary form.

*Solution :*

| BINARY   | DECIMAL |
|----------|---------|
| carry 11 | carry 1 |
| 10011    | 19      |
| +1001    | +9      |
| -----    | -----   |
| 11100    | 28      |

Note that while adding the first and the second column of the binary example, a carry is generated.

*Example 5.3.*

Add the binary numbers 100111 and 11011 in both decimal and binary form.



**Solution :**

|       | BINARY  | DECIMAL |
|-------|---------|---------|
| CARRY | 1111    | CARRY 1 |
|       | 100111  | 39      |
|       | + 11011 | + 27    |
|       | -----   | ----    |
|       | 1000010 | 66      |

In this example, we face a new situation ( $1+1+1$ ) brought about by the carry-over of 1 in the second column. This can also be handled using the same four rules for binary addition. The addition of three 1's can be broken up into two steps. First we add only two 1's giving 10 ( $1+1=10$ ). The third 1 is now added to this result to obtain 11 (a 1 sum with a 1 carry). So we conclude that  $1 + 1 + 1 = 1$  plus a carry of 1 to next higher column.

### SUBTRACTION

The principles of decimal subtraction can as well be applied to subtraction of numbers in other bases. It consists of two steps, which are repeated for each column of the numbers. The first step is to determine if it is necessary to borrow. If the subtrahend (the lower digit) is larger than the minuend (the upper digit), it is necessary to borrow from the column to the left. It is important to note here that the value borrowed depends upon the base of the number and is always the decimal equivalent of the base. Thus, in decimal, 10 is borrowed; in binary, 2 is borrowed; in octal, 8 is borrowed; in hexadecimal, 16 is borrowed. The second step is simply to subtract the lower value from the upper value. The complete table for binary subtraction is as follows :

|                                                |
|------------------------------------------------|
| $0 - 0 = 0$                                    |
| $1 - 0 = 1$                                    |
| $1 - 1 = 0$                                    |
| $0 - 1 = 1$ with a borrow from the next column |

Observe that the only case in which it is necessary to borrow is when 1 is subtracted from 0. The exact procedure is illustrated with the examples given below.

**Example 5.4.** Subtract  $01110_2$  from  $10101_2$ .

**Solution :**

|        |        |
|--------|--------|
| Borrow | { 12   |
|        | 0202   |
|        | 10101  |
|        | -01110 |
|        | -----  |
|        | 00111  |

In the first column, 0 is subtracted from 1. No borrow is required in this case and the result is 1. In the second column, we have to subtract 1 from 0. As seen in the table for binary subtraction, a borrow is necessary to perform this subtraction. So a 1 is borrowed from the third column which becomes 2 in the second column because the base is 2. A 1 in the 4s column is equal to 2 in the 2s column. Now, in the second column, we subtract 1 from 2 giving a result of 1. The borrow performed in the second column reduces the 1 in the third column to 0. So in the third column, once again we have to subtract 1 from 0 for which borrow is required. The fourth column contains a 0 and thus has nothing to borrow. Therefore, we have to borrow from the fifth column. Borrowing 1 from the fifth column gives 2 in the fourth column. A 1 in the 16s column equals 2 in the 8s column. Now the fourth column has something to borrow. When 1 of the 2 in the fourth column is borrowed, it becomes 2 in the third column. Now, in the third column, we subtract 1 from 2 giving a result of 1. The borrow performed in the third column reduces the 1 in the fifth column to 0 and the 2 in the fourth column to 1. Hence, subtraction of the fourth column is now 1 from 1, giving 0 and in the fifth column, subtraction is 0 from 0, giving 0. Thus the final result of subtraction is  $00111_2$ . The result may be verified by subtracting  $14_{10}$  ( $= 01110_2$ ) from  $21_{10}$  ( $10101_2$ ) which gives  $7_{10}$  ( $= 00111_2$ ).

**Example 5.5.** Subtract  $0111000_2$  from  $1011100_2$ .

**Solution :**

|        |           |
|--------|-----------|
| BORROW | 2         |
|        | 1011100   |
|        | - 0111000 |
|        | -----     |
|        | 0100100   |

The result may be verified by subtracting  $56_{10}$  ( $= 0111000_2$ ) from  $92_{10}$  ( $= 1011100_2$ ) which gives  $36_{10}$  ( $= 0100100_2$ ).

### Additive Method of Subtraction.

The direct method of subtraction using the borrow concept seems to be easiest when people perform subtraction with paper and pencil. However, when subtraction is implemented by means of digital components, this method is found to be less efficient than the additive method of subtraction. It may sound surprising that even subtraction is performed using an additive method. This method of subtraction by an additive approach is known as *complementary subtraction*.

In order to understand complementary subtraction, it is necessary to know what is meant by the complement of a



number. For a number which has  $n$  digits in it, a *complement* is defined as the difference between the number and the base raised to the  $n$ th power minus one. The definition is illustrated with the following examples :

**Example 5.6.** Find the complement of  $37_{10}$ .

**Solution :**

Since the number has 2 digits and the value of base is 10, so  $(\text{Base})^n - 1 = 10^2 - 1 = 99$

$$\text{Now } 99 - 37 = 62$$

Thus the complement of  $37_{10} = 62_{10}$

**Example 5.7.** Find the complement of  $6_8$ .

**Solution :**

Since the number has 1 digit and the value of base is 8, so  $(\text{Base})^n - 1 = 8^1 - 1 = 7_{10}$

$$\text{Also } 6_8 = 6_{10}$$

$$\text{Now } 7_{10} - 6_{10} = 1_{10} = 1_8$$

Thus the complement of  $6_8 = 1_8$ .

**Example 5.8.** Find the complement of  $10101_2$ .

**Solution :**

Since the number has 5 digits and the value of base is 2,

$$\text{So } (\text{Base})^n - 1 = 2^5 - 1 = 31_{10}$$

$$\text{Also } 10101_2 = 21_{10}$$

$$\text{Now } 31_{10} - 21_{10} = 10_{10} = 1010_2$$

Thus the complement of  $10101_2 = 01010_2$ .

Observe from Example 5.8 that in case of binary numbers, it is not necessary to go through the usual process of obtaining complement. Instead, when dealing with binary numbers, a quick way to obtain a number's complement is to transform all its 0's to 1's, and all its 1's to 0's. For example, the complement of  $1011010$  is  $0100101$ . Thus circuit for obtaining complement of a number in binary system can be easily designed at very less expense.

number. We will now see how subtraction is performed using the complementary method.

Subtraction by the complementary method involves the following three steps :

- Step 1 : Find the complement of the number you are subtracting (subtrahend);
- Step 2 : Add this to the number from which you are taking away (minuend);
- Step 3 : If there is a carry of 1, add it to obtain the result; if there is no carry, recompute the sum and attach a negative sign to obtain the result.

To illustrate the procedure, let us first consider few examples for decimal numbers.

**Example 5.9.** Subtract  $56_{10}$  from  $92_{10}$  using complementary method.

**Solution :**

- Step 1 : Complement of  $56_{10}$   
 $= 10^2 - 1 - 56$   
 $= 99 - 56$   
 $= 43_{10}$
- Step 2 :  $92$   
 $+ 43$  (complement of 56)  
 $----$   
 $135$   
 $|$   
 $|$
- Step 3 :  $--> 1$  (add the carry of 1)  
 $-----$
- Result  $= 36$

The result may be verified using the method of normal subtraction :  $92 - 56 = 36$ .

**Example 5.10.** Subtract  $35_{10}$  from  $18_{10}$  using complementary method.

**Solution :**

- Step 1 : Complement of  $35_{10}$   
 $= 10^2 - 1 - 35$   
 $= 99 - 35$



$$= 64_{10}$$

Step 2 :

$$\begin{array}{r} 18 \\ +64 \text{ (complement of 35)} \\ \hline \end{array}$$

$$82$$

Step 3 :

There is no carry. So recomplement the sum and attach a negative sign to obtain the result.

$$\begin{aligned} \text{Result} &= -(99 - 82) \\ &= -17 \end{aligned}$$

The result may be verified using the method of normal subtraction:

$$18 - 35 = -17$$

Let us re-work these examples using binary numbers.

**Example 5.11.** Subtract  $0111000_2$  ( $56_{10}$ ) from  $1011100_2$  ( $92_{10}$ ) using complementary method.

Solution :

$$\begin{array}{r} 1011100 \\ +1000111 \text{ (complement of } 0111000) \\ \hline 10100011 \\ | \\ \text{-----} \rightarrow 1 \text{ (add the carry of 1)} \\ \hline 0100100 \end{array}$$

$$\text{Result} = 0100100_2 = 36_{10}$$

Verify the result with the results of Example 5.5 and Example 5.9.

**Example 5.12.** Subtract  $100011_2$  ( $35_{10}$ ) from  $010010_2$  ( $18_{10}$ ) using complementary method.

Solution :

$$\begin{array}{r} 010010 \\ +011100 \text{ (complement of } 100011) \\ \hline 101110 \end{array}$$

As there is no carry, so we have to complement the sum and attach a negative sign to it. Hence,

$$\text{Result} = -010001_2 \text{ (complement of } 101110_2)$$

$$= -17_{10}$$

Verify the result with the result of Example 5.10.

**Example 5.13.** Subtract  $01110_2$  from  $10101_2$  using complementary method.

Solution :

$$\begin{array}{r} 10101 \\ +10001 \text{ (complement of } 01110) \\ \hline 100110 \\ | \\ \text{-----} \rightarrow 1 \text{ (add the carry of 1)} \\ \hline 00111 \end{array}$$

$$\text{Result} = 00111_2$$

Verify the result with the result of Example 5.4.

## MULTIPLICATION

Multiplication in the binary system also follows the same general rules as decimal multiplication. However, learning the binary multiplication is a trivial task because the table for binary multiplication is very short, with only four entries instead of the 100 necessary for decimal multiplication. The complete table for binary multiplication is as follows :

$$0 \times 0 = 0$$

$$0 \times 1 = 0$$

$$1 \times 0 = 0$$

$$1 \times 1 = 1$$

The method of binary multiplication is illustrated with the example given below. It is only necessary to copy the multiplicand if the digit in the multiplier is 1, and to copy all 0s if the digit in the multiplier is a 0. The ease with which each step of the operation is performed is apparent.

**Example 5.14.** Multiply the binary numbers 1010 and 1001.

|         |                 |
|---------|-----------------|
| 1010    | Multiplicand    |
| x1001   | Multiplier      |
| <hr/>   |                 |
| 1010    | Partial Product |
| 0000x   | Partial Product |
| 0000xx  | Partial Product |
| 1010xxx | Partial Product |
| <hr/>   |                 |
| 1011010 | Final Product   |



### 1011010 Final Product

Note that the multiplicand is simply copied when multiplier digit is 1 and when the multiplier digit is 0, the partial product is only a string of zero's. As in decimal multiplication, each partial product is shifted one place to the left from the previous partial product. Finally, all the partial products obtained in this manner are added according to the binary addition rules to obtain the final product.

In actual practice, whenever a 0 appears in the multiplier, a separate partial product consisting of a string of zeros need not be generated. Instead, only a left shift will do. As a result, Example 5.14 may be reduced to

$$\begin{array}{r}
 1010 \\
 \times 1001 \\
 \hline
 1010 \\
 1010ss \quad (s = \text{left shift}) \\
 \hline
 1011010
 \end{array}$$

A computer would also follow this procedure in performing multiplication. The result of this multiplication may be verified by multiplying  $10_{10}$  ( $1010_2$ ) by  $9_{10}$  ( $1001_2$ ) which produces a result of  $90_{10}$  ( $1011010_2$ ).

It may not be obvious how to handle the addition if the result of the multiplication gives columns with more than two 1s. They can be handled as pairs or by adjusting the column to which the carry is placed, as shown by Example 5.15.

### Additive Method of Multiplication.

Most of the computers perform multiplication operation by the way of addition only. This can be easily seen by considering an example, say  $4 \times 8$ . The result for this multiplication can be determined by evaluating, with necessary carry overs,  $8+8+8+8$ . That is, the result is obtained simply by adding the digit 8 four times. Similarly, the computer performs all multiplication operations in binary using the additive approach.

This idea of repeated addition may seem to be a longer way of doing things, but remember that the computer is well suited to carry out the operations at great speed. The internal circuit design of computer systems is

also simplified to a great extent by using this method of multiplication.

**Example 5.15.**

Multiply the binary numbers 1111 and 111.

**Solution :**

$$\begin{array}{r}
 1111 \\
 \times 111 \\
 \hline
 1111 \\
 1111 \\
 1111 \\
 \hline
 1101001
 \end{array}$$

Addition  
Handled  
As Pairs  
(Column 3)

$$\begin{array}{r}
 1 \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] \left. \begin{array}{l} \text{From} \\ \text{Problem} \end{array} \right\} \\
 1 \left[ \begin{array}{c} 1 \\ 1 \end{array} \right] \left. \begin{array}{l} \text{Carry} \\ \text{From} \\ \text{Col. 2} \end{array} \right\} \\
 \hline
 \uparrow 0 \\
 \text{Add two} \\
 \text{Carries} \\
 \text{to Col. 4}
 \end{array}$$

Addition  
Handled  
As Single  
Carry (Col. 3,

$$\begin{array}{r}
 \text{Sum} = 4 \\
 \left[ \begin{array}{c} 1 \\ 1 \\ 1 \\ 1 \end{array} \right] \left. \begin{array}{l} \text{From} \\ \text{Problem} \end{array} \right\} \\
 \left. \begin{array}{l} \text{Carry} \\ \text{From} \\ \text{Col. 2} \end{array} \right\} \\
 \hline
 100 \\
 \uparrow \uparrow \\
 \text{Carry to Col. 4} \\
 \text{Carry to Col. 5}
 \end{array}$$



## DIVISION

$$5 - 5 = 0$$

Binary division is, again, very simple. As in the decimal system (or in any other number system), division by zero is meaningless. Hence, the complete table for binary division is as follows :

$$0 / 1 = 0$$

$$1 / 1 = 1$$

The division process is performed in a manner similar to decimal division. The rules for binary division are :

1. Start from the left of the dividend.
2. Perform a series of subtractions in which the divisor is subtracted from the dividend.
3. If subtraction is possible, put a 1 in the quotient and subtract the divisor from the corresponding digits of dividend.
4. If subtraction is not possible (divisor greater than remainder), record a 0 in the quotient.
5. Bring down the next digit to add to the remainder digits. Proceed as before in a manner similar to long division

The method is illustrated by Example 5.16.

The result may be verified by dividing  $33_{10}$  ( $100001_2$ ) by  $6_{10}$  ( $110_2$ ) which gives a quotient of  $5_{10}$  ( $101_2$ ) and a remainder of  $3_{10}$  ( $11_2$ ).

**Additive Method of Division.**

Even division operation is performed inside most computers by the process of addition only. This may again sound surprising, but it is true. The computer performs the division operation essentially by repeating the complementary subtraction method. For example,  $35 / 5$  may be thought of as :

$$35 - 5 = 30$$

$$30 - 5 = 25$$

$$25 - 5 = 20$$

$$20 - 5 = 15$$

$$15 - 5 = 10$$

$$10 - 5 = 5$$

*Example 5.16.* Divide  $100001_2$  by  $110_2$ .  
*Solution :*

|            |               |                    |
|------------|---------------|--------------------|
|            | <b>0101</b>   | <b>(Quotient)</b>  |
| <b>110</b> | <b>100001</b> | <b>(Dividend)</b>  |
|            | <b>110</b>    | ← 1                |
|            | <b>1000</b>   | ← 2                |
|            | <b>110</b>    | ← 3                |
|            | <b>100</b>    | ← 4                |
|            | <b>110</b>    | ← 5                |
|            | <b>1001</b>   | ← 6                |
|            | <b>110</b>    | ← 7                |
|            | <b>11</b>     | <b>(Remainder)</b> |

- 1 Divisor greater than 100, so put 0 in quotient.
- 2 Add digit from dividend to group used above
- 3 Subtraction possible so put 1 in quotient
- 4 Remainder from subtraction plus digit from dividend
- 5 Divisor greater, so put 0 in quotient
- 6 Add digit from dividend to group
- 7 Subtraction possible, so put 1 in quotient



That is, the divisor is subtracted repeatedly from the dividend until the result of subtraction becomes less than or equal to zero. The total number of times subtraction was performed gives the value of the quotient. In this case, the value of quotient is 7 because the divisor (5) has been subtracted 7 times from the dividend (35) until the result of subtraction becomes zero. If the result of last subtraction is zero then there is no remainder for the division. But, if it is less than zero, then the last subtraction is ignored and the result of the previous subtraction is taken as the value of the remainder. In this case, the last subtraction operation is not counted for evaluating the value of the quotient. The process is illustrated below with an example.

**Example 5.17.** Divide  $33_{10}$  by  $6_{10}$  using the method of addition.

**Solution :**

$$33 - 6 = 27$$

$$27 - 6 = 21$$

$$21 - 6 = 15$$

$$15 - 6 = 9$$

$$9 - 6 = 3$$

$$3 - 6 = -3$$

Total number of subtractions = 6. Since the result of the last subtraction is less than zero, so

Quotient = 6 - 1 (ignore last subtraction) = 5

Remainder = 3 (result of previous subtraction)

Thus,  $33 / 6 = 5$  with a remainder 3.

Note that it has been assumed here that all the subtraction operations are carried out using the complementary subtraction method (additive method).

Once again, performing division inside a computer by the way of addition is desirable because the addition and complementation operations are easily performed in a computer and usually save the labour and expense of designing complicated circuits.

We have demonstrated how computer arithmetic is based on addition. Exactly how this simplifies matter can only be understood in the context of binary (not in decimal). The number of individual steps may indeed be

increased because all computer arithmetic is reduced to addition, but the computer can carry out binary additions at such great speed that this is not a disadvantage.

## QUESTIONS

1. Why have computers been designed to use the binary number system ?
2. Add the binary numbers 1011 and 101 in both decimal and binary form.
3. Add the binary numbers 1010110 and 1011010.
4. Add the binary numbers 10111 and 1011.
5. Find the complement of the following numbers :
  - a)  $495_{10}$     b)  $29_{10}$
  - c) 4,    d)  $C_{16}$
  - e) 2,    f) 32,
6. Find the complement of the following binary numbers :
  - a) 10    b) 101
  - c) 101101    d) 011011
  - e) 001101001110
7. Subtract  $0110111_2$  from  $1101110_2$ .
8. Subtract  $01010_2$  from  $10000_2$ .
9. Subtract  $011011_2$  from  $110111_2$ .
10. Subtract  $25_{10}$  from  $50_{10}$  using complementary method.
11. Subtract  $25_{10}$  from  $20_{10}$  using complementary method.
12. Subtract  $234_{10}$  from  $588_{10}$  using complementary method.
13. Subtract  $216_{10}$  from  $172_{10}$  using complementary method.
14. Subtract  $01010_2$  from  $10000_2$  using complementary method.



15. Subtract  $110111_2$  from  $101110_2$  using complementary method.
16. Subtract  $011011_2$  from  $110111_2$  using complementary method.
17. Subtract  $1111_2$  from  $1100_2$  using complementary method.
18. Multiply the binary numbers  $1100$  and  $1010$ .
19. Multiply the binary numbers  $01101$  and  $1001$ .
20. Multiply the binary numbers  $101111$  and  $111$ .
21. Divide  $11001_2$  by  $101_2$ .
22. Divide  $0110111_2$  by  $0111_2$ .
23. Briefly explain how multiplication and division operations are performed within a computer using additive approach.
24. What is the primary advantage of performing subtraction by the complementary method in digital computers ?
25. Discuss the advantages and disadvantages of performing the various arithmetic operations by the additive method in a digital computer.







CHAPTER 6

# 6. BOOLEAN ALGEBRA AND LOGIC CIRCUITS

In the previous chapters you have seen that computers normally use binary numbers. In this chapter, you will learn about an algebra that deals with the binary number system. This algebra, known as Boolean algebra, is very useful in designing logic circuits used by the processors of computer systems. In addition to this, you will also learn about the elementary logic gates that are used to build up circuits of different types to perform the necessary arithmetic operations. These logic gates are the building blocks of all the circuits in a computer. Finally, in this chapter, you will also learn how to use Boolean algebra to design simple logic circuits frequently used by the arithmetic logic unit of almost all computers.

## BOOLEAN ALGEBRA

In the mid-1800's, an algebra which simplified the representation and manipulation of propositional logic was

developed by the English mathematician, George Boole (1815-1864). It became known as Boolean algebra after its developer's name. Later, in the year 1938, Claude E. Shannon, a research assistant in the department of electrical engineering at the Massachusetts Institute of Technology, published a thesis entitled, "A Symbolic Analysis of Relay and Switching Circuits". In his thesis, he proposed the use of Boolean algebra in the design of relay switching circuits. The basic techniques described by Shannon were adopted almost universally for the design and analysis of switching circuits. Because of the analogous relationship between the action of relays and of modern electronic circuits, the same techniques which were developed for the design of relay circuits are still being used in the design of modern high-speed computers.

Boolean algebra provides an economical and straightforward approach to the design of relay and other



types of switching circuits. Just as an ordinary algebraic expression may be simplified by means of the basic theorems, the expression describing a given switching circuit network may also be reduced or simplified using Boolean algebra. Boolean algebra is now being used extensively in designing the circuitry used in computers. In short, a knowledge of Boolean algebra is must in the computing field.

## FUNDAMENTAL CONCEPTS OF BOOLEAN ALGEBRA

1. *Use of Binary digits.* In a normal algebraic expression, a variable can take any numerical value. For example, in the expression  $3A + 7B = C$ , we assume that A, B, and C may range through the entire field of real numbers.

Since Boolean algebra deals with the binary number system, the variables used in the Boolean equations may assume only two possible values (0 and 1). If an equation describing logical circuitry has several variables, it is still understood that each of the variables can assume only the values 0 or 1. For example, in the equation  $A + B = C$ , each of the variables A, B, and C may have only the values 0 or 1.

2. *Logical addition.* The symbol '+' is used for logical addition operator. It is also known as 'OR' operator. We can define the + symbol (OR operator) by listing all possible combinations of A and B and the resulting value of C in the equation  $A + B = C$ . It may be noted that since the variables A and B can have only two possible values (0 or 1) so only four ( $2^2$ ) combinations of inputs are possible as shown in Table 6.1. The resulting output values for each of the four input combinations are given in the table. Such a table is known as a *truth table*. Thus, Table 6.1 is truth table for the logical OR operator.

Observe that the result is 0 only when the value of both the input variables is 0. The result is 1 when any of the input variables is 1. Note that a result of 1 is also obtained when both the inputs A and B are 1. This is the reason why the + symbol does not have the "normal" meaning, but is a logical addition operator. This concept of logical addition may be extended to any number of variables. For example, in the equation  $A + B + C + D = E$ , even if A, B, C, and D all had the

value of 1, the sum of the values (the result E) would be 1 only. The equation  $A + B = C$  is normally read as "A or B equals C".

Table 6.1. Truth Table For Logical OR (+) Operator.

| INPUTS |   |   | OUTPUT |
|--------|---|---|--------|
| A      | + | B | = C    |
| 0      |   | 0 | 0      |
| 0      |   | 1 | 1      |
| 1      |   | 0 | 1      |
| 1      |   | 1 | 1      |

3. *Logical multiplication.* The symbol '.' is used for logical multiplication operator. It is also known as 'AND' operator. We can again define the . symbol (AND operator) by listing all possible combinations of A and B and the resulting value of C in the equation  $A . B = C$ . The truth table for logical AND operator is shown in Table 6.2. Observe from the truth table that the result C is equal to 1 only when both the input variables A and B are 1, otherwise it is 0. The equation  $A . B = C$  is normally read as "A and B equals C".

Table 6.2. Truth Table For Logical AND (.) Operator.

| INPUTS |   |   | OUTPUT |
|--------|---|---|--------|
| A      | . | B | = C    |
| 0      |   | 0 | 0      |
| 0      |   | 1 | 0      |
| 1      |   | 0 | 0      |
| 1      |   | 1 | 1      |

4. *Complementation.* The two operations defined so far (OR and AND) are binary operations because they define an operation on two variables. The complementation operation is a unary operation which is defined on a single variable.



The symbol  $\bar{\cdot}$  is normally used for complementation operator. It is also known as 'NOT' operator. Thus we write  $\bar{A}$ , meaning "take the complement of A," or  $\overline{(A + B)}$ , meaning "take the complement of  $A + B$ ." The complementation of a variable is the reverse of its value. Thus, if  $A = 0$  then  $\bar{A} = 1$  and if  $A = 1$  then  $\bar{A} = 0$ . The truth table for logical NOT ( $\bar{\cdot}$ ) operator is shown in Table 6.3.  $\bar{A}$  is read as "complement of A" or "not of A".

Table 6.3. Truth Table For Logical NOT ( $\bar{\cdot}$ ) Operator.

| INPUT | OUTPUT    |
|-------|-----------|
| A     | $\bar{A}$ |
| 0     | 1         |
| 1     | 0         |

5. *Operator precedence.* Does  $A + B \cdot C$  mean  $(A + B) \cdot C$  or  $A + (B \cdot C)$ ? The two generate different values for  $A = 1$ ,  $B = 0$ , and  $C = 0$ , for then we have  $(1 + 0) \cdot 0 = 0$  and  $1 + (0 \cdot 0) = 1$ , which differ. Hence it is necessary to define operator precedence in order to correctly evaluate Boolean expressions. The precedence of Boolean operators is as follows :

1. The expression is scanned from left to right.
2. Expressions enclosed within parentheses are evaluated first.
3. All complement (NOT) operations are performed next.
4. All  $\bar{\cdot}$  (AND) operations are performed after that.
5. Finally all  $+$  (OR) operations are performed in the end.

So according to this precedence rule,  $A + B \cdot C$  means  $A + (B \cdot C)$ . Similarly for the expression  $\bar{A} \cdot B$ , the complement of A and B are both evaluated first and the results are then ANDed. Again for the expression  $\overline{(A + B)}$ , the expression inside the parenthesis  $(A + B)$  is evaluated first and the result so obtained is then complemented.

## POSTULATES OF BOOLEAN ALGEBRA

### Postulate 1 :

- (a)  $A = 0$  if and only if A is not equal to 1
- (b)  $A = 1$  if and only if A is not equal to 0

### Postulate 2 :

- a)  $x + 0 = x$
- (b)  $x \cdot 1 = x$

### Postulate 3 : Commutative Law

- (a)  $x + y = y + x$
- (b)  $x \cdot y = y \cdot x$

### Postulate 4 : Associative Law

- (a)  $x + (y + z) = (x + y) + z$
- (b)  $x \cdot (y \cdot z) = (x \cdot y) \cdot z$

### Postulate 5 : Distributive Law

- (a)  $x \cdot (y + z) = x \cdot y + x \cdot z$
- (b)  $x + y \cdot z = (x + y) \cdot (x + z)$

### Postulate 6 :

- (a)  $x + \bar{x} = 1$
- (b)  $x \cdot \bar{x} = 0$

The postulates listed above are the basic axioms of the algebraic structure and need no proof. They are used to prove the theorems of Boolean algebra.

## THE PRINCIPLE OF DUALITY

In Boolean algebra, there is a precise duality between the operators  $\cdot$  (AND) &  $+$  (OR) and the digits 0 & 1. For instance, let us consider Table 6.4. We can see that the second row of the table is obtainable from the first row and vice - versa simply by interchanging  $\bar{\cdot}$  with  $\bar{\cdot}$  and  $\bar{0}$  with  $\bar{1}$ . This important property is known as the principle of duality in Boolean algebra.

The implication of this duality is that any theorem in Boolean algebra has dual obtainable by interchanging  $\bar{\cdot}$  with  $\bar{\cdot}$  and  $\bar{0}$  with  $\bar{1}$ .



with '1' and '0' with '1'. Thus if a particular theorem is proved, its dual theorem automatically holds and need not be proved separately.

Table 6.4. Illustrating The Principle Of Duality In Boolean Algebra.

|       | Column 1  | Column 2        | Column 3  |
|-------|-----------|-----------------|-----------|
| Row 1 | $1+1 = 1$ | $1+0 = 0+1 = 1$ | $0+0 = 0$ |
| Row 2 | $0.0 = 0$ | $0.1 = 1.0 = 0$ | $1.1 = 1$ |

### THEOREMS OF BOOLEAN ALGEBRA

Some of the important theorems of Boolean algebra are stated below along with their proof.

#### Theorem 1 (Idempotent law)

$$(a) \ x + x = x$$

$$(b) \ x \cdot x = x$$

##### Proof of (a)

L.H.S.

$$= x + x$$

$$= (x + x) \cdot 1 \quad \text{by postulate 2(b)}$$

$$= (x + x) \cdot (x + \bar{x}) \quad \text{by postulate 6(a)}$$

$$= x + x \cdot \bar{x} \quad \text{by postulate 5(b)}$$

$$= x + 0 \quad \text{by postulate 6(b)}$$

$$= x \quad \text{by postulate 2(a)}$$

$$= \text{R.H.S.}$$

##### Proof of (b)

L.H.S.

$$= x \cdot x$$

$$= x \cdot x + 0$$

$$= x \cdot x + x \cdot \bar{x} \quad \text{by postulate 6(b)}$$

$$= x \cdot (x + \bar{x}) \quad \text{by postulate 5(a)}$$

$$= x \cdot 1 \quad \text{by postulate 6(a)}$$

$$= x \quad \text{by postulate 2(b)}$$

$$= \text{R.H.S.}$$

Note that theorem 1(b) is the dual of theorem 1(a) and that each step of the proof in part (b) is the dual of part (a). Any dual theorem can be similarly derived from the proof of its corresponding pair. Hence from now onwards, the proof of part (a) only will be given. Interested readers can apply the principle of duality to the various steps of the proof of part (a) to obtain the proof of part (b) for any theorem.

#### Theorem 2

$$(a) \ x + 1 = 1$$

$$(b) \ x \cdot 0 = 0$$

##### Proof of (a);

L.H.S.

$$= x + 1$$

$$= (x+1) \cdot 1 \quad \text{by postulate 2(b)}$$

$$= (x+1) \cdot (x + \bar{x}) \quad \text{by postulate 6(a)}$$

$$= x + 1 \cdot \bar{x} \quad \text{by postulate 5(b)}$$

$$= x + \bar{x} \cdot 1 \quad \text{by postulate 3(b)}$$

$$= x + \bar{x} \quad \text{by postulate 2(b)}$$

$$= 1 \quad \text{by postulate 6(a)}$$

$$= \text{R.H.S.}$$

Proof of (b) holds by duality.

#### Theorem 3 (Absorption law)



(b)  $x \cdot (x+y) = x$

*Proof of (a)*

L.H.S.

$= x + x \cdot y$

$= x \cdot 1 + x \cdot y$  by postulate 2(b)

$= x \cdot (1+y)$  by postulate 5(a)

$= x \cdot (y+1)$  by postulate 3(a)

$= x \cdot 1$  by theorem 2(a)

$= x$  by postulate 2(b)

$= \text{R.H.S.}$

Proof of (b) holds by duality.

*Proof by the method of perfect induction.* The theorems of Boolean algebra can also be proved by means of truth tables. In a truth table, both sides of the relation are checked to yield identical results for all possible combinations of variables involved. In principle, it is possible to enumerate all possible combinations of the variables involved because Boolean algebra deals with variables that can have only two values. This method of proving theorems is called exhaustive enumeration or perfect induction.

Table 6.5. Truth Table For Proving Theorem 3(a) By The Method Of Perfect Induction.

| $x$ | $y$ | $x \cdot y$ | $x + x \cdot y$ |
|-----|-----|-------------|-----------------|
| 0   | 0   | 0           | 0               |
| 0   | 1   | 0           | 0               |
| 1   | 0   | 0           | 1               |
| 1   | 1   | 1           | 1               |

For example, Table 6.5 is a truth table for proving Theorem 3(a) by the method of perfect induction. Similarly

Table 6.6 proves Theorem 3(b) by the method of perfect induction.

Table 6.6. Truth Table For Proving Theorem 3(b) By The Method Of Perfect Induction.

| $x$ | $y$ | $x + y$ | $x \cdot (x + y)$ |
|-----|-----|---------|-------------------|
| 0   | 0   | 0       | 0                 |
| 0   | 1   | 1       | 0                 |
| 1   | 0   | 1       | 1                 |
| 1   | 1   | 1       | 1                 |

Theorem 4 (Involution Law)

$\overline{\overline{x}} = x$

Proof

Table 6.7 proves this theorem by the method of perfect induction.

Table 6.7. Truth Table For Proving Theorem 4 By The Method Of Perfect Induction.

| $x$ | $\overline{x}$ | $\overline{\overline{x}}$ |
|-----|----------------|---------------------------|
| 0   | 1              | 0                         |
| 1   | 0              | 1                         |

Note that Theorem 4 has no dual since it deals with the NOT operator which is unary operator.



## Theorem 5

(a)  $x \cdot (\bar{x} + y) = x \cdot y$

(b)  $x + \bar{x} \cdot y = x + y$

Proof of (a)

Table 6.8 proves this theorem by the method of perfect induction.

Table 6.8. Truth Table for Proving Theorem 5(a) By The Method Of Perfect Induction.

| x | y | $\bar{x}$ | $\bar{x} + y$ | $x \cdot (\bar{x} + y)$ | $x \cdot y$ |
|---|---|-----------|---------------|-------------------------|-------------|
| 0 | 0 | 1         | 1             | 0                       | 0           |
| 0 | 1 | 1         | 1             | 0                       | 0           |
| 1 | 0 | 0         | 0             | 0                       | 0           |
| 1 | 1 | 0         | 1             | 1                       | 1           |

Proof of (b)

Table 6.9 proves this theorem by the method of perfect induction.

Table 6.9. Truth Table For Proving Theorem 5(b) By The Method Of Perfect Induction.

| x | y | $\bar{x}$ | $\bar{x} \cdot y$ | $x + \bar{x} \cdot y$ | $x + y$ |
|---|---|-----------|-------------------|-----------------------|---------|
| 0 | 0 | 1         | 0                 | 0                     | 0       |
| 0 | 1 | 1         | 1                 | 1                     | 1       |
| 1 | 0 | 0         | 0                 | 1                     | 1       |
| 1 | 1 | 0         | 0                 | 1                     | 1       |

## Theorem 6 (De Morgan's law)

(a)  $\overline{x + y} = \bar{x} \cdot \bar{y}$

(b)  $\overline{x \cdot y} = \bar{x} + \bar{y}$

Proof of (a)

Table 6.10 proves this theorem by the method of perfect induction.

Table 6.10. Truth Table For Proving Theorem 6(a) By The Method Of Perfect Induction.

| x | y | $x + y$ | $\overline{x + y}$ | $\bar{x}$ | $\bar{y}$ | $\bar{x} \cdot \bar{y}$ |
|---|---|---------|--------------------|-----------|-----------|-------------------------|
| 0 | 0 | 0       | 1                  | 1         | 1         | 1                       |
| 0 | 1 | 1       | 0                  | 1         | 0         | 0                       |
| 1 | 0 | 1       | 0                  | 0         | 1         | 0                       |
| 1 | 1 | 1       | 0                  | 0         | 0         | 0                       |

Proof of (b)

Table 6.11 proves this theorem by the method of perfect induction.

Table 6.11. Truth Table For Proving Theorem 6(b) By The Method Of Perfect Induction.

| x | y | $x \cdot y$ | $\overline{x \cdot y}$ | $\bar{x}$ | $\bar{y}$ | $\bar{x} + \bar{y}$ |
|---|---|-------------|------------------------|-----------|-----------|---------------------|
| 0 | 0 | 0           | 1                      | 1         | 1         | 1                   |
| 0 | 1 | 0           | 1                      | 1         | 0         | 1                   |
| 1 | 0 | 0           | 1                      | 0         | 1         | 1                   |
| 1 | 1 | 1           | 0                      | 0         | 0         | 0                   |



Theorems 6(a) and 6(b) are important and very useful. They are known as De Morgan's laws. They can be extended to n variables as given below.

$$\overline{X_1 + X_2 + X_3 + \dots + X_n} = \overline{X_1} \cdot \overline{X_2} \cdot \overline{X_3} \dots \overline{X_n}$$

$$\overline{X_1 \cdot X_2 \cdot X_3 \dots X_n} = \overline{X_1} + \overline{X_2} + \overline{X_3} + \dots + \overline{X_n}$$

The basic Boolean identities are summarised in Table 6.12. It is suggested that the readers should become well conversant with the identities given in this table in order to use the algebra effectively.

Table 6.12. Summary Of Basic Boolean Identities.

| SL. NO. | IDENTITIES                                 | DUAL IDENTITIES                             |
|---------|--------------------------------------------|---------------------------------------------|
| 1.      | $A + 0 = A$                                | $A \cdot 1 = A$                             |
| 2.      | $A + 1 = 1$                                | $A \cdot 0 = 0$                             |
| 3.      | $A + \bar{A} = 1$                          | $A \cdot A = A$                             |
| 4.      | $A + \bar{A} = 1$                          | $A \cdot \bar{A} = 0$                       |
| 5.      | $\bar{\bar{A}} = A$                        |                                             |
| 6.      | $A + B = B + A$                            | $A \cdot B = B \cdot A$                     |
| 7.      | $(A + B) + C = A + (B + C)$                | $(A \cdot B) \cdot C = A \cdot (B \cdot C)$ |
| 8.      | $A \cdot (B + C) = A \cdot B + A \cdot C$  | $A + B \cdot C = (A + B) \cdot (A + C)$     |
| 9.      | $A + A \cdot B = A$                        | $A \cdot (A + B) = A$                       |
| 10.     | $A + \bar{A} \cdot B = A + B$              | $A \cdot (\bar{A} + B) = A \cdot B$         |
| 11.     | $\overline{A + B} = \bar{A} \cdot \bar{B}$ | $\overline{A \cdot B} = \bar{A} + \bar{B}$  |

the unary operator NOT, parentheses and equal sign. For a given value of the variables, the value of the function can be either 0 or 1. For example, consider the equation

$$W = X + \bar{Y} \cdot Z$$

Here the variable W is a function of X, Y and Z. This is written as  $W = f(X, Y, Z)$  and the right hand side of the equation is called an *expression*. The symbols X, Y and Z are referred to as *literals* of this function.

The above is an example of a Boolean function represented as an algebraic expression. A Boolean function may also be represented in the form of a truth table. The number of rows in the table will be equal to  $2^n$ , where n is the number of literals (binary variables) used in the function. The combinations of 0's and 1's for each row of this table is easily obtained from the binary numbers by counting from 0 to  $2^n - 1$ . For each row of the table, there is a value for the function equal to either 0 or 1 which is listed in separate column of the table. Such a truth table for the function  $W = X + \bar{Y} \cdot Z$  is shown in Table 6.13. Observe that there are eight ( $2^3$ ) possible distinct combinations for assigning bits to three variables. The column labeled W is either a 0 or a 1 for each of these combinations. The table shows that out of eight, there are five different combinations for which  $W = 1$ .

Table 6.13. Truth Table For The Boolean Function  $W = X + \bar{Y} \cdot Z$

| X | Y | Z | W |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

## BOOLEAN FUNCTIONS

A Boolean function is an expression formed with binary variables, the two binary operators OR and AND

The question now arises - is an algebraic expression for a given Boolean function unique? In other words, is it possible to find two algebraic expressions that specify the same Boolean function? The answer to this question is yes. As a matter of fact, the manipulation of Boolean algebra is



applied mostly to the problem of finding simpler expressions for a given expression. For example, let us consider the following two functions :

$$F_1 = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}$$

$$F_2 = x.\bar{y} + \bar{x}.z$$

Table 6.14. Truth Table For The Boolean Functions:

$$F_1 = \bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y} \text{ and}$$

$$F_2 = x.\bar{y} + \bar{x}.z$$

| x | y | z | F <sub>1</sub> | F <sub>2</sub> |
|---|---|---|----------------|----------------|
| 0 | 0 | 0 | 0              | 0              |
| 0 | 0 | 1 | 1              | 1              |
| 0 | 1 | 0 | 0              | 0              |
| 0 | 1 | 1 | 1              | 1              |
| 1 | 0 | 0 | 1              | 1              |
| 1 | 0 | 1 | 1              | 1              |
| 1 | 1 | 0 | 0              | 0              |
| 1 | 1 | 1 | 0              | 0              |

The representation of these two functions in the form of truth table is shown in Table 6.14. From the table we find that the function  $F_2$  is same as the function  $F_1$  since both have identical 0's and 1's for each combination of values of the three binary variables  $x$ ,  $y$  and  $z$ . In general, two functions of  $n$  binary variables are said to be equal if they have the same value for all possible  $2^n$  combinations of the  $n$  literals.

### MINIMIZATION OF BOOLEAN FUNCTIONS

When a Boolean function is implemented with logic gates (discussed later in this chapter), each literal in the function designates an input to a gate and each term is implemented with a gate. Thus for a given Boolean function, the minimization of the number of literals and the number of terms will result in a circuit with less equipments. For example, since functions  $F_1$  and  $F_2$  of Table 6.14 are equal Boolean functions, it is more economical to implement the  $F_2$  form than the  $F_1$  form because the  $F_2$  form contains fewer terms. To find simpler circuits, one must know how to manipulate Boolean functions to obtain equal

and simpler expressions. What constitutes the best form of Boolean function depends on the particular application. However, we will give consideration only to the criterion of equipment minimization which is achieved by literal minimization.

There are several methods used for minimizing the number of literals in a Boolean function. However, a discussion of all these methods is beyond the scope of this book. Hence here we will consider only the method of algebraic manipulations. Unfortunately, in this method there are no specific rules or guidelines to be followed that will guarantee the final answer. The only method available is cut-and-try procedure employing the postulates, the basic theorems, and any other manipulation method which becomes familiar with use. The following examples illustrate this procedure.

**Example 6.1.** Simplify the following Boolean functions to a minimum number of literals.

(a)  $x + \bar{x}.y$

(b)  $x . (\bar{x} + y)$

(c)  $\bar{x}.\bar{y}.z. + \bar{x}.y.z + x.\bar{y}$

(d)  $x.y + \bar{x}.z + y.z$

(e)  $(x+y) . (\bar{x}+z) . (y+z)$

**Solution :**

(a)

$$x + \bar{x}.y$$

$$= (x + \bar{x}).(x + y) \quad \text{by postulate 5(b)}$$

$$= 1.(x + y) \quad \text{by postulate 6(a)}$$

$$= (x + y).1 \quad \text{by postulate 3(b)}$$

$$= x + y \quad \text{by postulate 2(b)}$$

(b)

$$x.(\bar{x} + y)$$

$$= x.\bar{x} + x.y \quad \text{by postulate 5(a)}$$

$$= 0 + x.y \quad \text{by postulate 6(b)}$$



$$= x.y + 0 \quad \text{by postulate 3(a)}$$

$$= x.y \quad \text{by postulate 2(a)}$$

(c)

$$\bar{x}.\bar{y}.z + \bar{x}.y.z + x.\bar{y}$$

$$= \bar{x}.z.(\bar{y}+y) + x.\bar{y} \quad \text{by postulate 5(a)}$$

$$= \bar{x}.z.(y+\bar{y}) + x.\bar{y} \quad \text{by postulate 3(a)}$$

$$= \bar{x}.z.1 + x.\bar{y} \quad \text{by postulate 6(a)}$$

$$= \bar{x}.z + x.\bar{y} \quad \text{by postulate 2(b)}$$

(d)

$$x.y + \bar{x}.z + y.z$$

$$= x.y + \bar{x}.z + y.z.1 \quad \text{by postulate 2(b)}$$

$$= x.y + \bar{x}.z + y.z.(x+\bar{x}) \quad \text{by postulate 6(a)}$$

$$= x.y + \bar{x}.z + y.z.x + y.z.\bar{x} \quad \text{by postulate 5(a)}$$

$$= x.y + \bar{x}.z + x.y.z + \bar{x}.\bar{y}.z \quad \text{by postulate 3(b)}$$

$$= x.y.1 + \bar{x}.z + x.y.z + \bar{x}.\bar{y}.z \quad \text{by postulate 2(b)}$$

$$= x.y.1 + x.y.z + \bar{x}.z + \bar{x}.\bar{y}.z \quad \text{by postulate 3(a)}$$

$$= x.y.(1+z) + \bar{x}.z.(1+y) \quad \text{by postulate 5(a)}$$

$$= x.y.(z+1) + \bar{x}.z.(y+1) \quad \text{by postulate 3(a)}$$

$$= x.y.1 + \bar{x}.z.1 \quad \text{by theorem 2(a)}$$

$$= x.y + \bar{x}.z \quad \text{by postulate 2(b)}$$

(e)

$$(x+y) . (\bar{x}+z).(y+z)$$

$$= (x+y) . (\bar{x}+z) \quad \text{by duality from (d).}$$

are the dual of each other and use dual expressions in corresponding minimization steps. Function (c) shows the equality of the functions  $F_1$  and  $F_2$  of Table 6.14. Function (d) illustrates the fact that an increase in the number of literals sometimes leads to a final simpler expression. Observe that function (e) is the dual of function (d). Hence it is not minimized directly and can be easily derived from the dual of the steps used to derive function (d).

### COMPLEMENT OF A FUNCTION

The complement of a function  $F$  is  $\bar{F}$  and is obtained by interchanging 0's for 1's and 1's for 0's in the truth table that defines the function. For example Table 6.15 defines the function  $F = x.\bar{y} + \bar{x}.z$  and its complement  $\bar{F}$ .

Table 6.15. Truth Table For The Function  $F = x.\bar{y} + \bar{x}.z$  And Its Complement  $\bar{F}$ .

| x | y | z | F | $\bar{F}$ |
|---|---|---|---|-----------|
| 0 | 0 | 0 | 0 | 1         |
| 0 | 0 | 1 | 1 | 0         |
| 0 | 1 | 0 | 0 | 1         |
| 0 | 1 | 1 | 1 | 0         |
| 1 | 0 | 0 | 1 | 0         |
| 1 | 0 | 1 | 1 | 0         |
| 1 | 1 | 0 | 0 | 1         |
| 1 | 1 | 1 | 0 | 1         |

Algebraically, the complement of a function may be derived through De Morgan's theorems whose generalized forms are as follows :

$$\overline{A_1 + A_2 + A_3 + \dots + A_n} = \bar{A}_1 . \bar{A}_2 . \bar{A}_3 . \dots . \bar{A}_n$$

$$\overline{A_1 . A_2 . A_3 . \dots . A_n} = \bar{A}_1 + \bar{A}_2 + \bar{A}_3 + \dots + \bar{A}_n$$

These theorems state that the complement of a function is obtained by interchanging the OR and the AND operators and complementing each literal. The method is illustrated below through an example.

Example 6.2.

Find the complement of the following functions :

(a)  $F_1 = \bar{x}.y.\bar{z} + \bar{x}.\bar{y}.z$

(b)  $F_2 = x.(y.z + y.\bar{z})$



**Solution :**

Applying De Morgan's theorems as many times as necessary, the complements are obtained as follows :

(a)

$$\begin{aligned}\overline{F_1} &= \overline{\overline{x} \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot z} \\ &= (\overline{\overline{x} \cdot \overline{y} \cdot \overline{z}}) \cdot (\overline{\overline{x} \cdot \overline{y} \cdot z}) \\ &= (\overline{\overline{x}} + \overline{\overline{y}} + \overline{\overline{z}}) \cdot (\overline{\overline{x}} + \overline{\overline{y}} + \overline{z}) \\ &= (x + \overline{y} + z) \cdot (x + \overline{y} + \overline{z})\end{aligned}$$

(b)

$$\begin{aligned}\overline{F_2} &= \overline{x \cdot (\overline{y} \cdot \overline{z} + y \cdot z)} \\ &= \overline{x} + \overline{(\overline{y} \cdot \overline{z} + y \cdot z)} \\ &= \overline{x} + (\overline{\overline{y} \cdot \overline{z}}) \cdot (\overline{y \cdot z}) \\ &= \overline{x} + (\overline{\overline{y}} + \overline{\overline{z}}) \cdot (\overline{y} + \overline{z}) \\ &= \overline{x} + (y + z) \cdot (\overline{y} + \overline{z})\end{aligned}$$

A simpler procedure for deriving the complement of a function is to take the dual of the function and then complement each literal. This method follows from the generalized De Morgan's theorems. Remember that the dual of a function is obtained by interchanging OR and AND operators and 0's and 1's. The method is illustrated below with the help of an example.

**Example 6.3.** Find the complement of the functions  $F_1$  and  $F_2$  of Example 6.2. by taking their dual and complementing each literal.

**Solution:**

(a)

$$F_1 = \overline{x} \cdot \overline{y} \cdot \overline{z} + \overline{x} \cdot \overline{y} \cdot z$$

The dual of  $F_1$  is:  $(\overline{x} + y + \overline{z}) \cdot (\overline{x} + \overline{y} + z)$

Complementing each literal we get

$$\overline{F_1} = (x + \overline{y} + z) \cdot (x + \overline{y} + \overline{z})$$

(b)

$$F_2 = x \cdot (\overline{y} \cdot \overline{z} + y \cdot z)$$

The dual of  $F_2$  is:  $x + (\overline{y} + \overline{z}) \cdot (y + z)$

Complementing each literal we get

$$\overline{F_2} = \overline{x} + (y + z) \cdot (\overline{y} + \overline{z})$$

## CANONICAL FORMS FOR BOOLEAN FUNCTIONS

**Minterms and Maxterms.** A binary variable may appear either in its normal form ( $x$ ) or in its complement form ( $\overline{x}$ ). Now consider two binary variables  $x$  and  $y$  combined with an AND operation. Since each variable may appear in either form, there are four possible combinations :

$$\overline{x} \cdot \overline{y}, \overline{x} \cdot y, x \cdot \overline{y}, x \cdot y$$

Each of these four AND terms is called a *minterm* or a *standard product*.

In a similar manner,  $n$  variables can be combined to form  $2^n$  minterms. The  $2^n$  different minterms may be determined by a method similar to the one shown in Table 6.16 below for three variables. The binary numbers from 0 to  $2^n - 1$  are listed under the  $n$  variables. Each minterm is obtained from an AND term of the  $n$  variables, with each variable being primed if the corresponding bit of the binary number is 0 and unprimed if a 1.

Table 6.16. Minterms And Maxterms For Three Variables.

| VARIABLES |   |   | MINTERMS                                             |             | MAXTERMS                                     |             |
|-----------|---|---|------------------------------------------------------|-------------|----------------------------------------------|-------------|
| X         | Y | Z | TERM                                                 | DESIGNATION | TERM                                         | DESIGNATION |
| 0         | 0 | 0 | $\overline{x} \cdot \overline{y} \cdot \overline{z}$ | $m_0$       | $x + y + z$                                  | $M_0$       |
| 0         | 0 | 1 | $\overline{x} \cdot \overline{y} \cdot z$            | $m_1$       | $x + y + \overline{z}$                       | $M_1$       |
| 0         | 1 | 0 | $\overline{x} \cdot y \cdot \overline{z}$            | $m_2$       | $x + \overline{y} + z$                       | $M_2$       |
| 0         | 1 | 1 | $\overline{x} \cdot y \cdot z$                       | $m_3$       | $x + \overline{y} + \overline{z}$            | $M_3$       |
| 1         | 0 | 0 | $x \cdot \overline{y} \cdot \overline{z}$            | $m_4$       | $\overline{x} + y + z$                       | $M_4$       |
| 1         | 0 | 1 | $x \cdot \overline{y} \cdot z$                       | $m_5$       | $\overline{x} + y + \overline{z}$            | $M_5$       |
| 1         | 1 | 0 | $x \cdot y \cdot \overline{z}$                       | $m_6$       | $\overline{x} + \overline{y} + z$            | $M_6$       |
| 1         | 1 | 1 | $x \cdot y \cdot z$                                  | $m_7$       | $\overline{x} + \overline{y} + \overline{z}$ | $M_7$       |

A symbol for each minterm is also shown in the table and is of the form  $m_j$ , where  $j$  denotes the decimal equivalent of the binary number of the minterm designated.

In a similar fashion,  $n$  variables forming an OR term, with each variable being primed or unprimed, provide  $2^n$  possible combinations called *maxterms* or *standard sums*.

The eight maxterms for three variables, together



with their symbolic designation, are listed in Table 6.16. Any  $2^n$  maxterms for  $n$  variables may be determined similarly. Each maxterm is obtained from an OR term of the  $n$  variables, with each variable being unprimed if the corresponding bit is 0 and primed if it is a 1.

Note that each maxterm is the complement of its corresponding minterm and vice-versa.

### Sum-of-Products

A sum-of-products expression is a product term (minterm) or several product terms (minterms) logically added (ORed) together.

For example, the expression  $x\bar{y} + \bar{x}y$  is a sum-of-products expression. The following are all sum-of-products expressions:

$$x$$

$$x+y$$

$$x + y.z$$

$$x.y + z$$

$$\bar{x}\bar{y} + x\bar{y}\bar{z}$$

The following steps are followed to express a Boolean function in its sum-of-products form :

1. Construct a truth table for the given Boolean function.
2. Form a minterm for each combination of the variables which produces a 1 in the function.
3. The desired expression is the sum (OR) of all the minterms obtained in step 2.

For example, in case of function  $F_1$  of Table 6.17, the following three combinations of the variables produce a 1:

001, 100 and 111

Their corresponding minterms are

$$\bar{x}\bar{y}.z, x\bar{y}\bar{z} \text{ and } x.y.z$$

Hence, taking the sum (OR) of all these minterms, the function  $F_1$  can be expressed in its sum-of-products form as:

$$F_1 = \bar{x}\bar{y}.z + x\bar{y}\bar{z} + x.y.z$$

or

$$F_1 = m_1 + m_4 + m_7$$

Similarly, it may be easily verified that the function  $F_2$  of Table 6.17 can be expressed in its sum-of-products form as:

$$F_2 = \bar{x}.y.z + x\bar{y}.z + x.y\bar{z} + x.y.z$$

or

$$F_2 = m_3 + m_5 + m_6 + m_7$$

Table 6.17. Truth Table For Functions  $F_1$  And  $F_2$ .

| x | y | z | $F_1$ | $F_2$ |
|---|---|---|-------|-------|
| 0 | 0 | 0 | 0     | 0     |
| 0 | 0 | 1 | 1     | 0     |
| 0 | 1 | 0 | 0     | 0     |
| 0 | 1 | 1 | 0     | 1     |
| 1 | 0 | 0 | 1     | 0     |
| 1 | 0 | 1 | 0     | 1     |
| 1 | 1 | 0 | 0     | 1     |
| 1 | 1 | 1 | 1     | 1     |

It is sometimes convenient to express a Boolean function in its sum-of-products form. If not in this form, it can be made so by first expanding the expression into a sum of AND terms. Each term is then inspected to see if it contains all the variables. If it misses one or more variables, it is ANDed with an expression of the form  $(x+\bar{x})$ , where  $x$  is one of the missing variables. The following example clarifies this procedure.

**Example 6.4.** Express the Boolean function  $F = A + \bar{B}.C$  in the sum-of-minterms (products) form.

**Solution :**

The function has three variables  $A$ ,  $B$  and  $C$ . The first term  $A$  is missing two variables, therefore;

$$A = A(B+\bar{B}) = AB + A\bar{B}$$



This is still missing one variable, so

$$\begin{aligned} A &= A.B.(C+\bar{C}) + A.\bar{B}.(C+\bar{C}) \\ &= A.B.C + A.B.\bar{C} + A.\bar{B}.C + A.\bar{B}.\bar{C} \end{aligned}$$

The second term  $\bar{B}.C$  is missing one variable, therefore

$$\begin{aligned} \bar{B}.C &= \bar{B}.C.(A+\bar{A}) \\ &= A.\bar{B}.C + \bar{A}.\bar{B}.C \end{aligned}$$

So by combining all the terms we get

$$F = A.B.C + A.B.\bar{C} + A.\bar{B}.C + A.\bar{B}.\bar{C} + A.\bar{B}.C + \bar{A}.\bar{B}.C$$

But in the above expression, the term  $A.\bar{B}.C$  appears twice and according to theorem 1(a) we have  $x+x=x$ . Hence it is possible to remove one of the them. Rearranging the minterms in ascending order, we finally obtain :

$$\begin{aligned} F &= \bar{A}.\bar{B}.C + A.\bar{B}.\bar{C} + A.\bar{B}.C + A.B.\bar{C} + A.B.C \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

It is sometimes convenient to express the Boolean function, when in its sum-of-minterms, in the following short notation:

$$F(A,B,C) = \sum(1,4,5,6,7)$$

The summation symbol  $\sum$  stands for the ORing of terms. The numbers following it are the minterms of the function. And finally, the letters in parentheses with F form a list of the variables in the order taken when the minterm is converted to an AND term.

### Product-of-Sums

A product-of-sums expression is a sum term (maxterm) or several sum terms (maxterms) logically multiplied (ANDed) together. For example, the expression  $(\bar{x}+y).(x+\bar{y})$  is a product of sums expression. The following are all product-of-sums expressions:

x

$$(\bar{x}+y)$$

$$(\bar{x}+\bar{y}).z$$

$$(x+\bar{y}).(\bar{x}+y).(\bar{x}+\bar{y})$$

$$(x+y).(\bar{x}+y+z)$$

The following steps are followed to express a Boolean function in its product-of-sums form:

1. Construct a truth table for the given Boolean function.
2. Form a maxterm for each combination of the variables which produce a 0 in the function.
3. The desired expression is the product (AND) of all the maxterms obtained in step 2.

For example in case of function  $F_1$  of Table 6.17, the following five combinations of the variables produce a 0: 000, 010, 011, 101, and 110

Their corresponding maxterms are :

$$(x+y+z), (x+\bar{y}+z), (x+\bar{y}+\bar{z}), (\bar{x}+y+\bar{z}), \text{ and } (\bar{x}+\bar{y}+z)$$

Hence, taking the product (AND) of all these maxterms, the function  $F_1$  can be expressed in its product-of-sums form as:

$$\begin{aligned} F_1 &= (x+y+z).(x+\bar{y}+z).(x+\bar{y}+\bar{z}).(\bar{x}+y+\bar{z}).(\bar{x}+\bar{y}+z) \\ \text{or} \\ F_1 &= M_0 . M_2 . M_3 . M_5 . M_6 \end{aligned}$$

Similarly, it may be easily verified that the function  $F_2$  of Table 6.17 can be expressed in its product-of-sums form as :

$$\begin{aligned} F_2 &= (x+y+z).(x+y+\bar{z}).(x+\bar{y}+z).(\bar{x}+y+z) \\ \text{or} \\ F_2 &= M_0 . M_1 . M_2 . M_4 \end{aligned}$$

In order to express a Boolean function in its product-of-sums form, it must first be brought into a form of OR terms. This may be done by using the distributive law :

$$x + y.z = (x+y).(x+z)$$

Then any missing variable (say x) in each OR term is ORed with the form  $x.\bar{x}$ . This procedure is clarified by the following example :

**Example 6.5.** Express the Boolean function

$$\begin{aligned} F &= x.y + \bar{x}.z \\ &\text{in the product-of-maxterms (sums) form.} \end{aligned}$$

**Solution:**



the distributive law:

$$\begin{aligned} F &= x.y + \bar{x}.z \\ &= (x.y + \bar{x}). (x.y + z) \\ &= (x + \bar{x}). (y + \bar{x}). (x + z). (y + z) \\ &= (\bar{x} + y). (x + z). (y + z) \end{aligned}$$

The function has three variables  $x$ ,  $y$  and  $z$ . Each OR term is missing one variable, therefore :

$$\overline{x+y} = \overline{x+y+z} \cdot \overline{z} = (\overline{x+y+z}) \cdot (\overline{x+y+z})$$

$$x+z = x+z + y.\bar{y} = (x+z+y).(x+z+\bar{y})$$

$$y+z = x.\bar{x} + y+z = (x+y+z).(\bar{x}+y+z)$$

Combining all the terms and removing those that appear more than once, we finally obtain :

$$F = (x+y+z).(x+\bar{y}+z).(\bar{x}+y+z).(\bar{x}+y+\bar{z})$$

$$= M_0 . M_2 . M_4 . M_5$$

A convenient way to express this function is as follows:

$$F(x,y,z) = \pi(0,2,4,5)$$

The product symbol  $\Pi$  denotes the ANDing of maxterms. The numbers following it are the maxterms of the function.

The sum-of-products and the product-of-sums form of Boolean expressions are known as *standard forms*. One prime reason for liking the sum-of-products or the product-of-sums expressions is their straightforward conversion to very nice gating networks which are more desirable from most implementation points of view. In their purest, nicest form they go into two-level networks, which are networks for which the longest path through which the signal must pass from input to output is two gates.

## CONVERSION BETWEEN CANONICAL FORMS

The complement of a function expressed as the sum-of-minterms equals the sum-of-minterms missing from the original function. This is because the original function is expressed by those minterms that make the function equal to 1, while its complement is a 1 for those minterms for which the function is a 0. For example, the function

$$F(A,B,C) = \sum(1,4,5,6,7)$$

has a complement that can be expressed as :

$$\begin{aligned}\overline{F}(A,B,C) &= \Sigma(0,2,3) \\ &= m_0 + m_2 + m_3\end{aligned}$$

Now, if we take the complement of  $\overline{F}$ , by De Morgan's theorem we obtain  $F$  back in a different form :

$$\begin{aligned} F &= \overline{m_0 + m_2 + m_3} \\ &= \overline{m_0} \cdot \overline{m_2} \cdot \overline{m_3} \\ &= M_0 \cdot M_2 \cdot M_3 \\ &= \Pi(0,2,3) \end{aligned}$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 6.16. From the table, it is clear that the following relation holds true :

$$\overline{m}_j = M_j$$

That is, the maxterm with subscript  $j$  is a complement of the minterm with the same subscript  $j$ , and vice versa.

The last example has demonstrated the conversion between a function expressed in sum-of-minterms and its equivalent in product-of-maxterms. A similar argument will show that the conversion between the product-of-maxterms and the sum-of-minterms is similar. We now state a general conversion procedure:

"To convert from one canonical form to another, interchange the symbol and list those numbers missing from the original form."

For example, the function

$$F(x,y,z) = \pi(0,2,4,5)$$

is expressed in the product-of-maxterms form. Its conversion to sum-of-minterms is :

$$F(x,y,z) = \sum(1,3,6,7)$$

Note that in order to find the missing terms, one must realize that the total number of minterms or maxterms is always  $2^n$ , where  $n$  is the number of binary variables in the function.

## LOGIC GATES

All operations within a computer are carried out by means of combinations of signals passing through standard blocks of built-in circuits that are known as logic gates. In other words, a *logic gate* is simply an electronic circuit



which operates on one or more input signals to produce standard output signals. These logic gates are the building blocks of all the circuits in a computer.

Computer circuits are built up using combinations of different types of logic gates to perform the necessary operation. There are several types of gates, but we shall consider here only some of the most important ones. These are sufficient to introduce the concept of circuit design using logic gates.

### AND GATE

An AND gate is the physical realization of the logical multiplication (AND) operation. That is, it is an electronic circuit that generates an output signal of 1 only if all input signals are also 1.

To have a conceptual idea, let us consider the case of Figure 6.1. Here two switches A and B are connected in series. It is obvious that the input current will reach the output point only when both the switches are in the on(1) state. There will be no output (output = 0) if either one or both the switches are in the off(0) state. So, two or more switches connected in series behave as an AND gate.

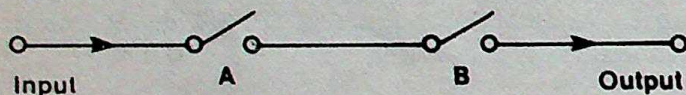


Figure 6.1. Two or more switches connected in series behave as an AND gate.

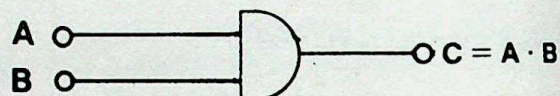
The behaviour of a logic gate, that is the state of its output signal depending on the various combinations of input signals, is conveniently described by means of a truth table. The truth table and the block diagram symbol for an AND gate for two input signals are shown in Figure 6.2. Since there are only two inputs (A & B), so only four ( $2^2$ ) combinations of inputs are possible. Also observe from the truth table that an output of 1 is obtained only when both the inputs are in 1 state, otherwise it is 0.

### OR GATE

An OR gate is the physical realization of the logical addition (OR) operation. That is, it is an electronic circuit that generates an output signal of 1 if any of the input signals is also 1.

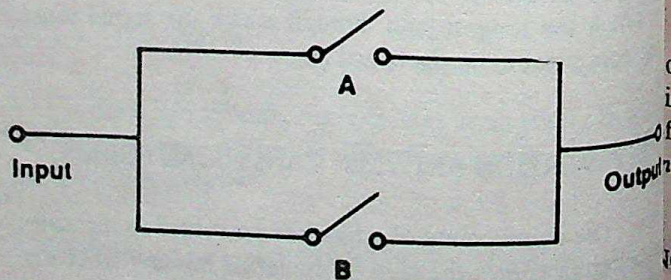
Two or more switches connected in parallel behave as an OR gate. It can be seen from Figure 6.3. that the input current will reach the output point when any one of the two switches are in the on(1) state. There will be no output only when both the switches (A & B) are in the off(0) state.

The truth table and the block diagram symbol for an OR gate for two input signals are shown in Figure 6.4. Observe that an output of 1 is obtained when any of the input signals is 1. Output is 0 only when both the inputs are 0.

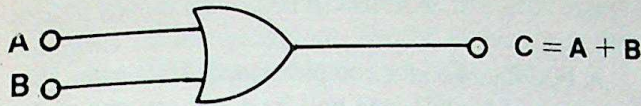


| INPUTS |   | OUTPUT          |
|--------|---|-----------------|
| A      | B | $C = A \cdot B$ |
| 0      | 0 | 0               |
| 0      | 1 | 0               |
| 1      | 0 | 0               |
| 1      | 1 | 1               |

Figure 6.2. Block diagram symbol and truth table for an AND gate.







| INPUTS |   | OUTPUT      |
|--------|---|-------------|
| A      | B | $C = A + B$ |
| 0      | 0 | 0           |
| 0      | 1 | 1           |
| 1      | 0 | 1           |
| 1      | 1 | 1           |

Figure 6.4. Block diagram symbol and truth table for an OR gate.

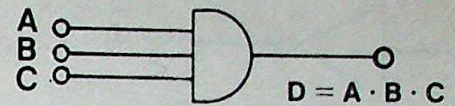
Just as the + and . operations could be extended to several variables using the associative law, AND gates and OR gates can have more than two inputs. Figure 6.5 shows three input AND and OR gates and the table of all input combinations for each. As might be hoped, the output of the AND gate with inputs A, B, and C is a 1 only if A and B and C are 1, i.e., when all three of the inputs are 1, so that we write the output as  $A \cdot B \cdot C$ . Similarly, the OR gate with inputs A, B, and C has a 1 output if A or B or C is a 1, so that we can write  $A + B + C$  for its output.

The above argument can be extended. A four-input AND gate has a 1 output only when all four inputs are 1, and a four-input OR gate has a 1 output when any of its inputs is a 1.

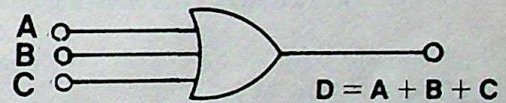
### NOT GATE

A NOT gate is the physical realization of the complementation operation. That is, it is an electronic circuit that generates an output signal which is the reverse of the input signal. A NOT gate is also known as an *inverter* because it inverts the input.

The truth table and the block diagram symbol for a NOT gate are shown in Figure 6.6. Recall that the complementation operation is unary operation which is defined on a single variable. Hence a NOT gate always has single input. Figure 6.6 shows also that connecting two NOT gates in series gives an output equal to the input, and this is the gating counterpart to the law of the double complementation,  $A = A$ .



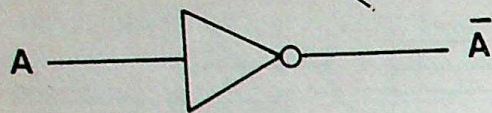
| INPUTS |   |   | OUTPUT |
|--------|---|---|--------|
| A      | B | C | D      |
| 0      | 0 | 0 | 0      |
| 0      | 0 | 1 | 0      |
| 0      | 1 | 0 | 0      |
| 0      | 1 | 1 | 0      |
| 1      | 0 | 0 | 0      |
| 1      | 0 | 1 | 0      |
| 1      | 1 | 0 | 0      |
| 1      | 1 | 1 | 1      |



| INPUTS |   |   | OUTPUT |
|--------|---|---|--------|
| A      | B | C | D      |
| 0      | 0 | 0 | 0      |
| 0      | 0 | 1 | 1      |
| 0      | 1 | 0 | 1      |
| 0      | 1 | 1 | 1      |
| 1      | 0 | 0 | 1      |
| 1      | 0 | 1 | 1      |
| 1      | 1 | 0 | 1      |
| 1      | 1 | 1 | 1      |

Figure 6.5. Three input AND and OR gates.





| INPUT |  | OUTPUT    |  |
|-------|--|-----------|--|
| A     |  | $\bar{A}$ |  |
| 0     |  | 1         |  |
| 1     |  | 0         |  |

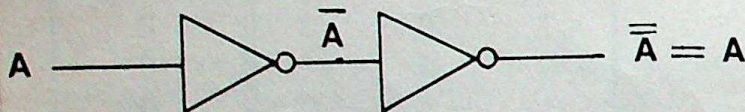
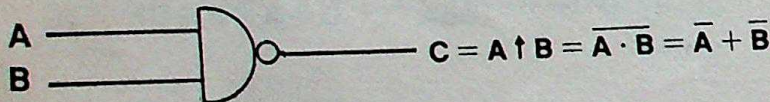


Figure 6.6. (a) Block diagram symbol and truth table for a NOT gate.

(b) Two NOT gates in series.



| INPUTS |   | OUTPUT |
|--------|---|--------|
| A      | B |        |
| 0      | 0 | 1      |
| 0      | 1 | 1      |
| 1      | 0 | 1      |
| 1      | 1 | 0      |

Figure 6.7. Block diagram symbol and truth table for a NAND gate.

## NAND GATE

A NAND gate is a complemented AND gate. That is, the output of NAND gate will be a 1 if any one of the inputs is a 0 and will be a 0 only when all the inputs are 1.

The truth table and the block diagram symbol for a NAND gate are shown in Figure 6.7. The symbol ' $\uparrow$ ' is usually used to represent a NAND operation in boolean expressions. Thus,  $A \uparrow B = \bar{A} \cdot \bar{B} = \bar{A+B}$ .

The operation of a NAND gate can be analysed using the equivalent block diagram circuit shown in Figure 6.8, which has an AND gate followed by a NOT gate. For inputs A and B, the output of the AND gate will be  $A \cdot B$  which is fed as input to the NOT gate. So the complement of  $A \cdot B$  will be  $\overline{A \cdot B}$  which is equal to  $\bar{A} + \bar{B}$  or  $A \uparrow B$ . In fact, small circle on the output of the NAND gate (see Figure 6.7) represents complementation. The NAND gate can then be seen to be an AND gate followed by a NOT gate.

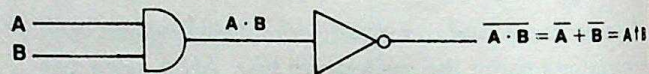


Figure 6.8. NAND gate realization with an AND gate and a NOT gate.

Multiple-input NAND gates can be analysed similarly. A three-input NAND gate with inputs A, B, and C will have an output equal to  $\bar{A} \cdot \bar{B} \cdot \bar{C}$  or  $\bar{A+B+C}$ , which says that the output will be a 1 if any of the inputs is a 0 and will be a 0 only when all three inputs are 1.

## NOR GATE

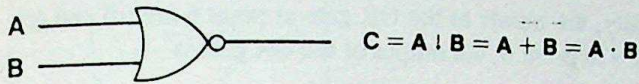
A NOR gate is a complemented OR gate. That is, the output of a NOR gate will be a 1 only when all inputs are 0 and it will be a 0 if any input represents a 1.

The truth table and the block diagram symbol for a NOR gate are shown in Figure 6.9. The symbol ' $\downarrow$ ' is usually used to represent a NOR operation in Boolean expressions. Thus,  $A \downarrow B = \overline{A+B} = \bar{A} \cdot \bar{B}$ .

The operation of a NOR gate can be analysed using the equivalent block diagram circuit shown in Figure 6.10, which has an OR gate followed by a NOT gate. For inputs A and B, the output of the OR gate will be  $A + B$  which is fed as input to the NOT gate. So the complement of  $A + B$  will be  $\overline{A+B}$  which is equal to  $\bar{A} \cdot \bar{B}$  or  $A \downarrow B$ . In fact, the



small circle on the output of the NOR gate (see Figure 6.9) represents complementation. The NOR gate can then be seen to be an OR gate followed by a NOT gate.



| INPUTS |   | OUTPUT                 |
|--------|---|------------------------|
| A      | B | $C = \overline{A + B}$ |
| 0      | 0 | 1                      |
| 0      | 1 | 0                      |
| 1      | 0 | 0                      |
| 1      | 1 | 0                      |

Figure 6.9. Block diagram symbol and truth table for a NOR gate.

Multiple input NOR gates can be analysed similarly. A three-input NOR gate with inputs A, B, and C will have an output equal to  $\overline{A + B + C}$  or  $\overline{A} \cdot \overline{B} \cdot \overline{C}$ , which says that the output will be a 1 only when all the three inputs are 0 and it will be a 0 if any of the three inputs is a 1.

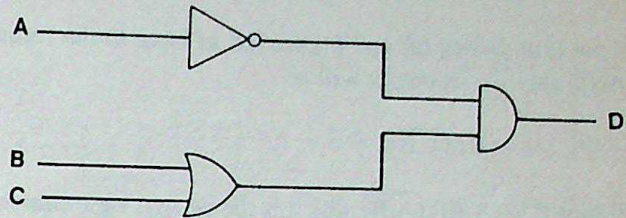


Figure 6.10. NOR gate realization with an OR gate and a NOT gate.

## LOGIC CIRCUITS

The logic gates described in the previous section are seldom used by themselves but are used in combinations. They are interconnected to form gating, or logic, networks which are known as combinational logic circuits. For these logic circuits, the Boolean algebra expression can be derived by systematically progressing from input to output on the gates. Few examples are given below.

**Example 6.6.** Find the Boolean expression for the output of the logic circuit given below.



**Solution :**

Input A is fed to the NOT gate whose output will be  $\overline{A}$ .

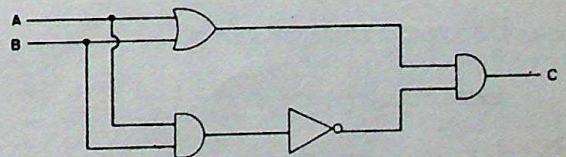
Inputs B and C are fed to the OR gate whose output will be  $B + C$ .

Now these two outputs ( $\overline{A}$  and  $B + C$ ) are fed as input to the AND gate. So the output produced by the AND gate will be  $\overline{A} \cdot (B + C)$

$$\text{Hence } D = \overline{A} \cdot (B + C)$$

which is the required Boolean expression for the output of the given logic circuit.

**Example 6.7.** Find the logic equation for the output produced by the logic circuit given below.



**Solution :**

The output of the OR gate is

$$A + B \text{ ----- (a)}$$

The output of the first AND gate is

$$A \cdot B \text{ ----- (b)}$$



Since the expression (b) is fed as input to the NOT gate. So the output of the NOT gate is

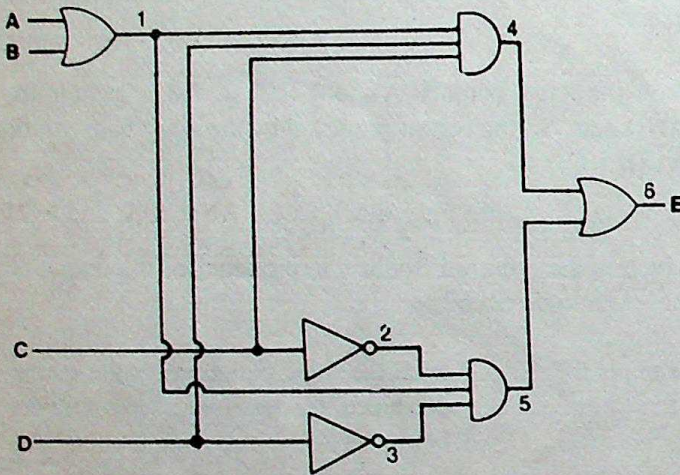
$$\overline{A.B} \text{ ----- (c)}$$

Now expressions (a) and (c) are fed as input to the second AND gate. So its output will be

$$(A+B).(\overline{A.B})$$

Hence  $C = (A+B).(\overline{A.B})$  which is the desired logic equation for the output produced by the given logic circuit.

**Example 6.8.** Find the Boolean expression for the output of the logic circuit given below.



**Solution :**

At point 1, the output of the OR gate is

$$A+B \text{ ----- (a)}$$

At point 2, the output of the NOT gate is

$$\overline{C} \text{ ----- (b)}$$

At point 3, the output of the NOT gate is

$$\overline{D} \text{ ----- (c)}$$

The inputs to the AND gate at point 4 are (A+B), C, and D. Hence at point 4, the output of the AND gate is

$$(A+B).C.D \text{ ---- (d)}$$

The inputs to the AND gate at point 5 are (A+B),  $\overline{C}$ , and  $\overline{D}$ . Hence at point 5, the output of the AND gate is

$$(A+B).\overline{C}.\overline{D} \text{ ---- (e)}$$

Finally, the inputs to the OR gate at point 6 are (d) and (e). Hence at point 6, the output of the OR gate is

$$(A+B).C.D. + (A+B).\overline{C}.\overline{D}.$$

$$\text{So } E = (A+B).C.D. + (A+B).\overline{C}.\overline{D}$$

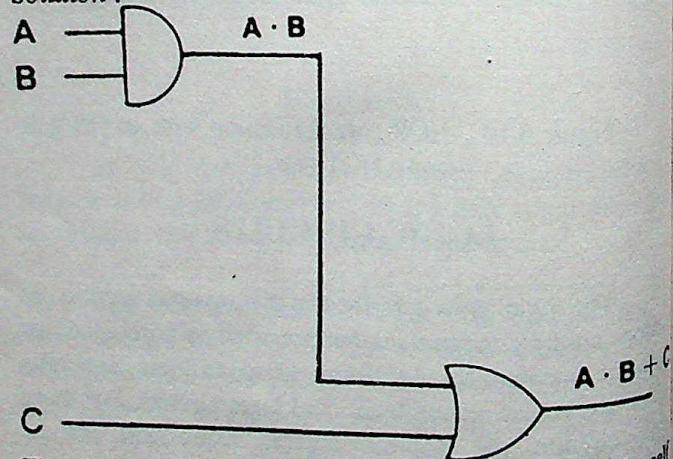
which is the required Boolean expression for the output of the given logic circuit.

### CONVERTING EXPRESSIONS TO LOGIC CIRCUITS

We have just now considered few examples that illustrate the method of deriving Boolean expression for a given logic circuit. The reverse problem of constructing a logic circuit for a given Boolean expression is also not difficult. The three logic gates - AND, OR, and NOT are said to be logically complete because any Boolean expression may be realized using only these three gates. The method of constructing logic circuits for Boolean expressions using only these three gates is illustrated below with the help of some examples.

**Example 6.9.** Construct a logic circuit for the Boolean expression  $A.B + C$ .

**Solution :**

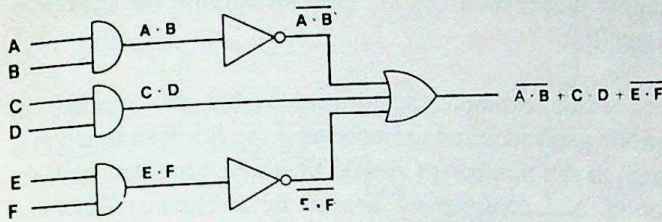


The desired logic circuit is shown above which is self explanatory.

**Example 6.10.** Construct a logic circuit for the Boolean expression  $\overline{A.B} + C.D + \overline{E.F}$



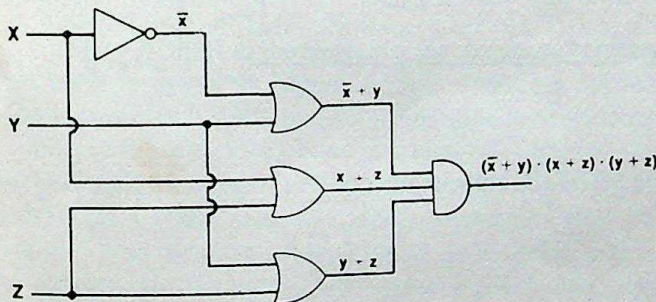
*Solution :*



The desired logic circuit is shown above which is self explanatory.

**Example 6.11.** Construct a logic circuit for the Boolean expression  $(\bar{x}+y) \cdot (x+z) \cdot (y+z)$

*Solution :*



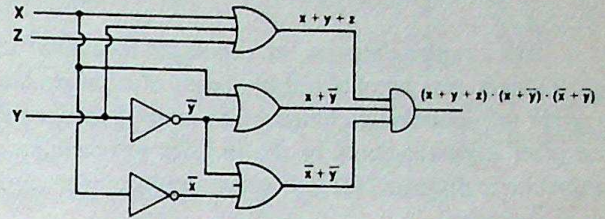
The desired logic circuit is shown above which is self explanatory.

### THE UNIVERSAL NAND GATE

We have seen that AND, OR, and NOT gates are logically complete in the sense that any Boolean function may be realized using these three gates. However, the NAND gate, which was introduced in the previous section, is said to be universal gate because it is alone sufficient to implement any Boolean function.

**Example 6.12:** Construct a logic circuit for the Boolean expression  $(x+y+z) \cdot (x+y) \cdot (x+z)$

*Solution :*



The desired logic circuit is shown above which is self explanatory.

To show that any Boolean function can be implemented with the sole use of NAND gates, we need only show that the logical operations AND, OR, and NOT can be implemented with NAND gates. This is shown in Figure 6.11 below.

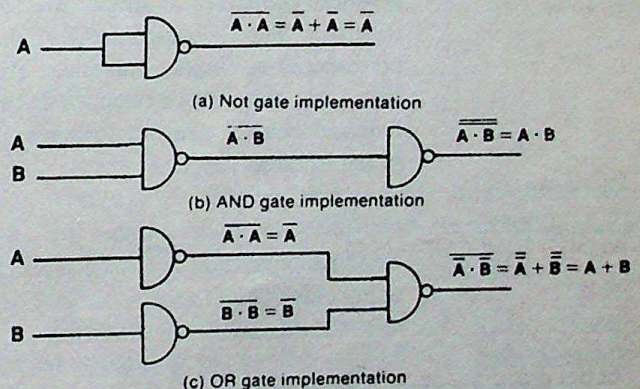


Figure 6.11. Implementation of NOT, AND, and OR gates by NAND gates.

A NOT operation is obtained from a one-input NAND gate. Thus we find that a single-input NAND gate behaves as an inverter.

The AND operation requires two NAND gates. The first one produces the inverted AND and the second one



are removed since they form double inversion which has no meaning. The inverter connected to input A is removed and the input variable is changed from A to  $\bar{A}$ . The result is the NAND logic diagram shown in Figure 6.12(c), with the number inside each NAND gate identifying the gate from Figure(b).

For the OR operation, the normal inputs A and B are first complemented using two single input NAND gates. Now the complemented variables are fed as input to another NAND gate which produces the normal ORed output.

The implementation of Boolean functions with NAND gates may be obtained by means of a simple block diagram manipulation technique. The method requires that two other logic diagrams be drawn prior to obtaining the NAND logic diagram. The following steps are to be carried out in sequence :

This example demonstrates that the number of NAND gates required to implement the Boolean function is equal to the number of AND/OR gates, provided both the normal and complement inputs are available. Otherwise inverters must be used to generate any required complemented inputs.

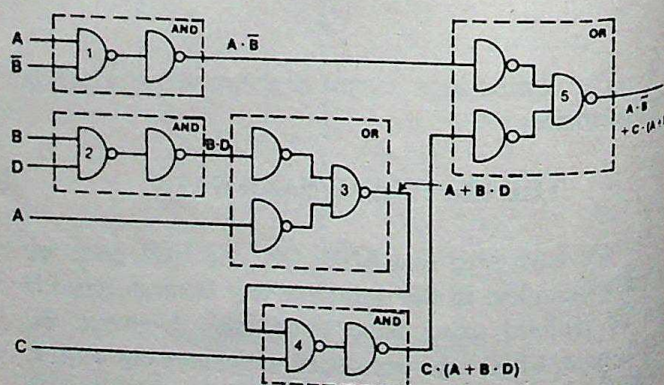
- Step 1: From the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal ( $A$ ) and complement ( $\bar{A}$ ) inputs are available.
- Step 2: Draw a second logic diagram with the equivalent NAND logic substituted for each AND, OR, and NOT gate.
- Step 3: Remove any two cascaded inverters from the diagram since double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable. The new logic diagram so obtained is the required NAND gate implementation of the Boolean function.

(a) AND/OR implementation.

**Example 6.13.** Construct a logic circuit for the Boolean expression  $A\overline{B} + C.(A+B.D)$  using only NAND gates.

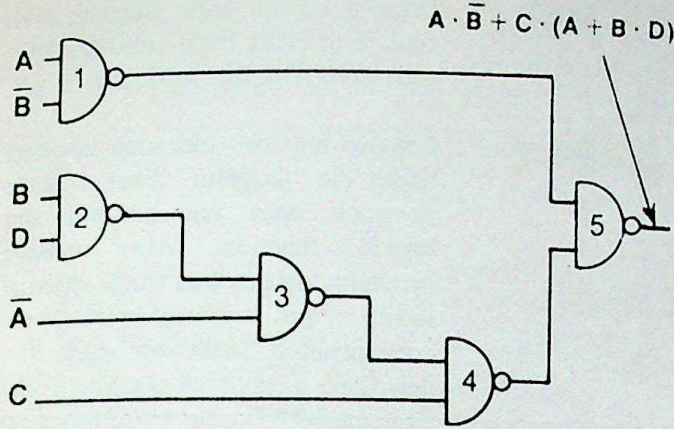
**Solution :**

The AND/OR implementation for the given Boolean expression is drawn in Figure 6.12(a). Now each AND gate is substituted by a NAND gate followed by an inverter and each OR gate is substituted by two input inverters followed by a NAND gate. Thus each AND gate is substituted by two NAND gates and each OR gate is substituted by three NAND gates. The logic diagram so obtained is shown in Figure 6.12(b). Note that Figure 6.12(b) has seven inverters (single input NAND gates) and five two-input NAND gates. Each two-input NAND gate has a number inside the gate symbol for identification purpose. Pairs of inverters connected in cascade (from each AND box to each OR box)



(b) Substituting equivalent NAND functions.





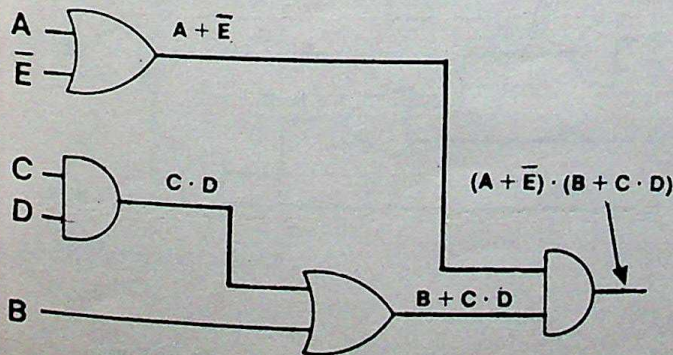
(c) NAND implementation.

Figure 6.12. Step-by-step NAND implementation for the Boolean expression of Example 6.13.

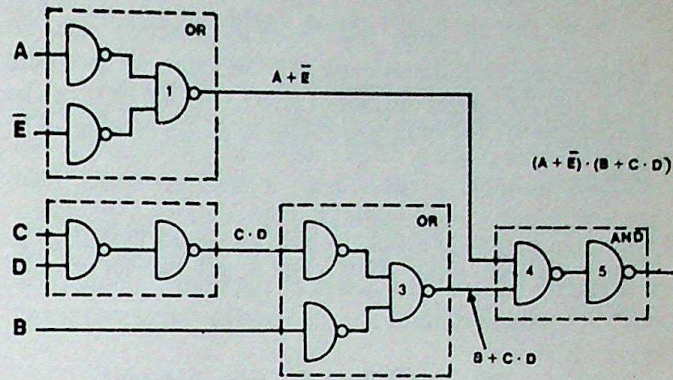
**Example 6.14.** Construct a logic circuit for the Boolean expression  $(A + \bar{E}) \cdot (B + C \cdot D)$  using only NAND gates.

**Solution :**

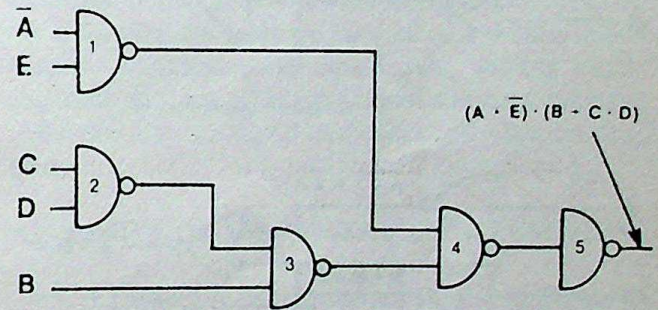
The AND/OR implementation for the given Boolean expression is drawn in Figure 6.13(a). Now the NAND equivalent of each AND and each OR gate is substituted resulting in Figure 6.13(b). Note that Figure 6.13(b) has six inverters (single input NAND gates) and four two-input NAND gates. One pair of cascaded inverters may be removed. Also the three external inputs A, B, and  $\bar{E}$ , which go directly to inverters, are complemented and the corresponding inverters are removed. The final NAND gate implementation so obtained is shown in Figure 6.13(c). The number inside each NAND gate of Figure 6.13(c) corresponds to the NAND gate of Figure 6.13(b) having the same number.



(a) AND/OR implementation.



(b) Substituting equivalent NAND functions.



(c) NAND implementation.

Figure 6.13. Step-by-step NAND implementation for the Boolean expression of Example 6.14.

For this example, the number of NAND gates required is equal to the number of AND/OR gates plus an additional inverter at the output (NAND gate number 5). In general, the number of NAND gates required to implement a Boolean function equals the number of AND/OR gates, except for an occasional inverter. This is true only when both normal and complemented inputs are available because the conversion forces certain input variables to be complemented.

### THE UNIVERSAL NOR GATE

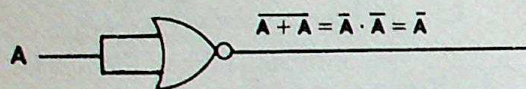
The NOR function is the dual of the NAND function. For this reason, all procedures and rules for NOR logic form a dual of the corresponding procedures and rules for NAND logic.



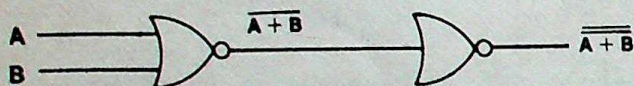
developed from NAND logic. Like the NAND gate, the NOR gate is also universal because it is alone sufficient to implement any Boolean function.

To show that any Boolean function can be implemented with the sole use of NOR gates, we need only show that the logical operations AND, OR, and NOT can be implemented with NOR gates. This is shown in Figure 6.14 below.

The NOT operation is obtained from a one-input NOR gate. Thus, a single input NOR gate is yet another inverter circuit.



(a) Not gate implementation



(b) OR gate implementation

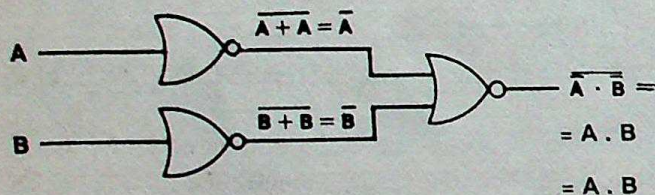


Figure 6.14. Implementation of NOT, OR and AND gates by NOR gates.

The OR operation requires two NOR gates. The first one produces the inverted OR and the second one being a single input NOT gate, acts as an inverter to obtain the normal OR output.

The AND operation is achieved through a NOR gate with additional inverters in each input.

Similar to the NAND logic diagram, the implementation of Boolean functions with NOR gates may be obtained by carrying out the following steps in sequence

- Step 1: For the given algebraic expression, draw the logic diagram with AND, OR, and NOT gates. Assume that both the normal (A) and complement ( $\bar{A}$ ) inputs are available.

- Step 2: Draw a second logic diagram with equivalent NOR logic substituted for each AND, OR, and NOT gate.

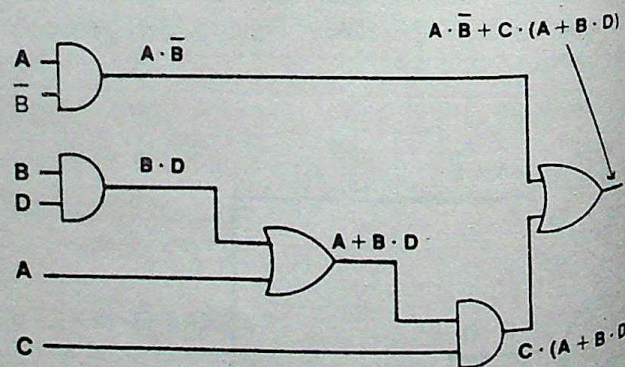
- Step 3: Remove any two cascaded inverters from the diagram since double inversion does not perform any logical function. Also remove inverters connected to single external inputs and complement the corresponding input variable. The new logic diagram so obtained is the required NAND gate implementation of the given Boolean function.

Example 6.15.

Construct a logic diagram for the Boolean expression  $A\bar{B} + C.(A+B.D)$  using only NOR gates.

Solution :

The AND/OR implementation for the given Boolean expression is shown in Figure 6.15(a). Now each OR gate is substituted by a NOR gate followed by an inverter and each AND gate is substituted by two input inverters followed by a NOR gate. Thus each OR gate is substituted by two NOR gates and each AND gate is substituted by three NOR gates. The logic diagram so obtained is shown in Figure 6.15(b). Note that Figure 6.15(b) has eight inverters (single input NOR gates) and five two-input NOR gates. One pair of cascaded inverters (from the OR box to the AND box) may be removed. Also the five external inputs  $A, \bar{B}, B, D$  and  $C$  which go directly to inverters, are complemented and the corresponding inverters are removed.



(a) AND/OR implementation.



The final NOR gate implementation so obtained is shown in Figure 6.15(c). The number inside each NOR gate of Figure 6.15(c) corresponds to the NOR gate of Figure 6.15(b) having the same number.

The number of NOR gates in this example equals the number of AND/OR gates plus an additional inverter in the output (NOR gate number 6). In general, the number of NOR gates required to implement a Boolean function equals the number of AND/OR gates, except for an occasional inverter. This is true only if both normal and complement inputs are available because the conversion forces certain input variables to be complemented.

Combinational circuits are more frequently constructed with NAND or NOR gates than with AND, OR and NOT gates. NAND and NOR gates are more popular than the AND and OR gates because NAND and NOR gates are easily constructed with transistor circuits and Boolean functions can be easily implemented with them. Moreover, NAND and NOR gates are superior to AND and OR gates from the hardware point of view, as they supply outputs that maintain the signal value without loss of amplitude. OR and AND gates sometimes need amplitude restoration after the signal travels through a few levels of gates.

### EXCLUSIVE-OR AND EQUIVALENCE FUNCTIONS

Exclusive-or and equivalence, denoted by  $\oplus$  and  $\odot$  respectively, are binary operations that perform the following Boolean functions:

$$A \oplus B = A\bar{B} + \bar{A}B$$

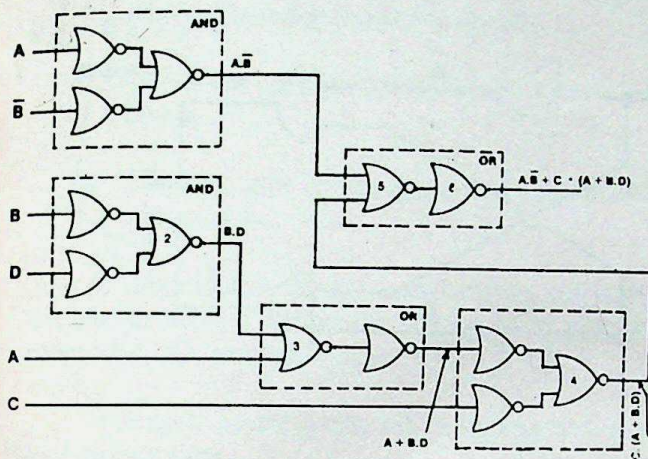
$$A \odot B = A.B + \bar{A}\bar{B}$$

The truth table and the block diagram symbol for the exclusive-or and the equivalence operations are shown in Figure 6.16 and Figure 6.17 respectively. Observe that the two operations are the complement of each other. Each is commutative and associative. Because of these two properties, a function of three or more variables can be expressed without parentheses as follows:

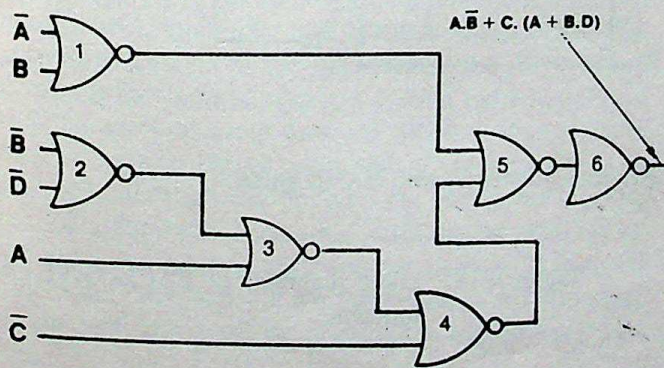
$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

$$= A \oplus B \oplus C$$

The exclusive-or and equivalence operations have many excellent characteristics as candidates for logic gates but are expensive to construct with physical components. They are available as standard logic gates in IC packages but are usually constructed internally with other standard



(b) Substituting equivalent NOR functions.

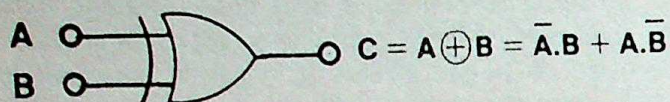


(c) NOR implementation.

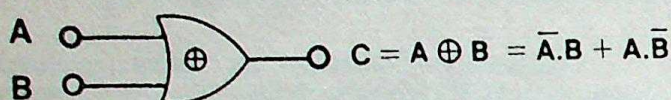
Figure 6.15. Step-by-step NOR implementation for the Boolean expression of Example 6.15.



gates. For example, Figure 6.18(a) shows the implementation of a two-input exclusive-or function with AND, OR and NOT gates. Figure 6.18(b) shows its implementation with NAND gates.

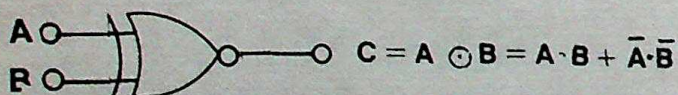


or



| INPUTS |   | OUTPUT           |
|--------|---|------------------|
| A      | B | $C = A \oplus B$ |
| 0      | 0 | 0                |
| 0      | 1 | 1                |
| 1      | 0 | 1                |
| 1      | 1 | 0                |

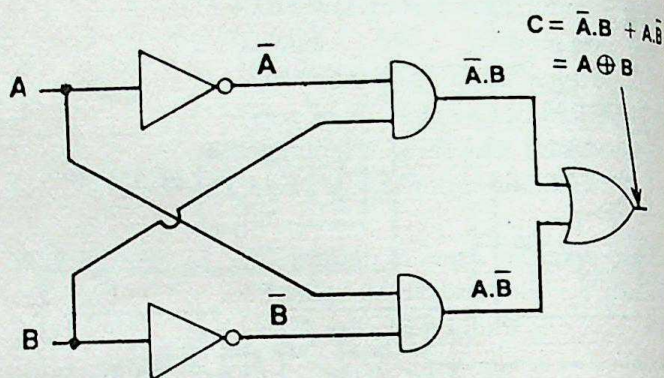
Figure 6.16. Block diagram symbol and truth table for an EXCLUSIVE-OR operation.



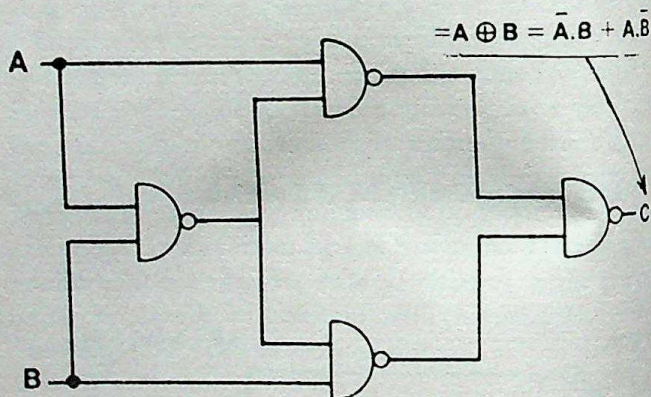
| INPUTS |   | OUTPUT          |
|--------|---|-----------------|
| A      | B | $C = A \odot B$ |
| 0      | 0 | 1               |
| 0      | 1 | 0               |
| 1      | 0 | 0               |
| 1      | 1 | 1               |

Figure 6.17. Block diagram symbol and truth table for an EQUIVALENCE operation.

Only a limited number of Boolean functions can be expressed exclusively in terms of exclusive-or or equivalence operations. Nevertheless, these functions emerge quite often during the design of digital systems. The two functions are particularly useful in arithmetic operations and error detection and correction.



(a) Implementation with AND/OR/NOT gates.



(b) Implementation with NAND gates.

Figure 6.18. Logic diagrams of EXCLUSIVE-OR function.

## DESIGN OF COMBINATIONAL CIRCUITS

The design of combinational circuits starts from the verbal outline of the problem and ends in a logic circuit diagram.

The procedure involves the following steps:

1. State the given problem completely and exactly.







The half-adder is limited in the sense that it can add only two single bits. Although it generates a carry for the next higher pair of significant bits, it cannot accept a carry generated from the previous pair of lower significant bits. A full-adder solves this problem.

### DESIGN OF FULL-ADDER

A full-adder forms the arithmetic sum of three input bits. Thus it consists of three inputs and two outputs. Two of the input variables (A and B) represent the augend and the addend bits and the third input variable (D) represents the carry from the previous lower significant position. Two outputs are necessary because the sum of three binary digits ranges in value from 0 to 3, and binary 2 and 3 need two digits. These two outputs are designated by the symbols S (for sum) and C (for carry). The truth-table of Figure 6.21 exactly defines the function of full-adder. The 1's and 0's for the output variables are determined from the arithmetic sum of the three input variables. When all input variables are 0, the output is 0 for both C and S. The S output is equal to 1 when only one input is equal to 1 or when all three inputs are equal to 1. The C output is 1 if two or three inputs are equal to 1.

| INPUTS |   |   | OUTPUTS |   |
|--------|---|---|---------|---|
| A      | B | D | C       | S |
| 0      | 0 | 0 | 0       | 0 |
| 0      | 0 | 1 | 0       | 1 |
| 0      | 1 | 0 | 0       | 1 |
| 0      | 1 | 1 | 1       | 0 |
| 1      | 0 | 0 | 0       | 1 |
| 1      | 0 | 1 | 1       | 0 |
| 1      | 1 | 0 | 1       | 0 |
| 1      | 1 | 1 | 1       | 1 |

Figure 6.21 Truth table for a full-adder.

The sum-of-products expressions for the two outputs can be directly obtained from the truth table and is given below :

$$S = \bar{A}.\bar{B}.D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D} + A.B.D$$

$$C = \bar{A}.B.D + A.\bar{B}.D + A.B.\bar{D} + A.B.D$$

Although the expression for S cannot be simplified, it is possible to simplify the expression for C as follows :

$$C = \bar{A}.B.D + A.\bar{B}.D + A.B.\bar{D} + A.B.D$$

$$= \bar{A}.B.D + A.\bar{B}.D + A.B.\bar{D} + A.B.D + A.B.D$$

A.B.D (since  $x + x = x$ )

$$= (\bar{A}.B.D + A.B.D) + (A.\bar{B}.D + A.B.D) + (A.B.\bar{D} + A.B.D)$$

$$= (A + \bar{A}).B.D + (B + \bar{B}).A.D + (D + \bar{D}).A.B$$

$$= B.D + A.D + A.B \text{ (since } x + \bar{x} = 1 \text{)}$$

$$= A.B + A.D + B.D$$

Hence, finally we obtain the following expressions for the two outputs :

$$S = \bar{A}.\bar{B}.D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D} + A.B.D$$

$$C = A.B + A.D + B.D$$

The logic circuit diagram to implement this is shown in Figure 6.22.

A full-adder can also be implemented with two half-adders and one OR gate as shown in Figure 6.23.

The S output from the second half-adder is the exclusive-OR of D and the output of the first half-adder giving :

$$S = (\bar{A}.B + A.\bar{B}).D + (\bar{A}.B + A.\bar{B}).\bar{D}$$

$$= (\bar{A}.B + A.\bar{B}).D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D}$$

$$= (\bar{A} + B).(\bar{A} + \bar{B}).D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D}$$

$$= (A + \bar{B}).(\bar{A} + B).D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D}$$

$$= (A.\bar{A} + A.B + \bar{A}.\bar{B} + B.\bar{B}).D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D}$$

$$= (A.B + \bar{A}.\bar{B}).D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D}$$

$$= A.B.D + \bar{A}.\bar{B}.D + \bar{A}.B.\bar{D} + A.\bar{B}.\bar{D}$$

$$= \bar{A}.B.D + \bar{A}.\bar{B}.\bar{D} + A.\bar{B}.\bar{D} + A.B.D$$



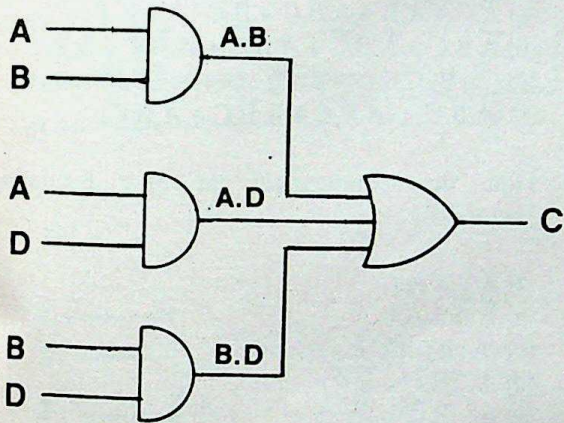
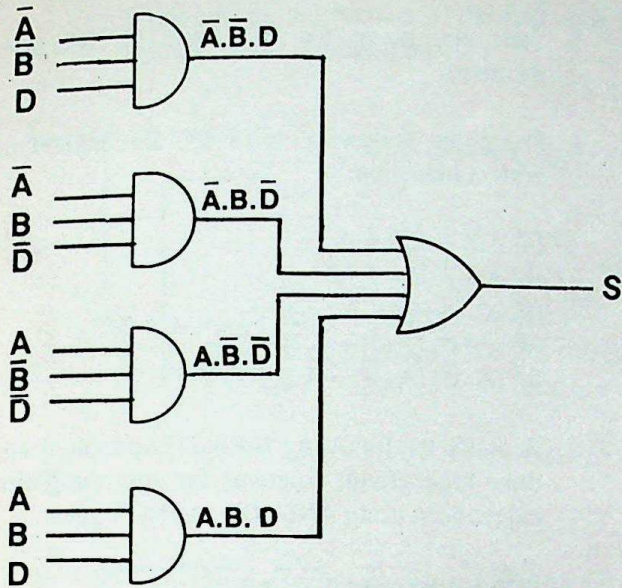


Figure 6.22. Logic Circuit diagram for a full-adder.

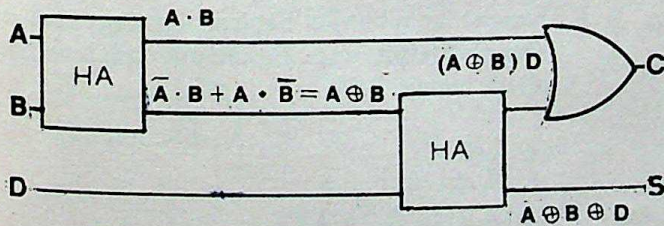


Figure 6.23. Implementation of full-adder with two half-adders and one OR gate.

And we have the carry output

$$\begin{aligned} C &= (\bar{A}.B + A.\bar{B}).D + A.B \\ &= \bar{A}.B.D + A.\bar{B}.D + A.B.(D + \bar{D}) \\ &= \bar{A}.B.D + A.\bar{B}.D + A.B.D + A.B.\bar{D} \end{aligned}$$

This can be simplified as before to

$$C = A.B + A.D + B.D$$

## A PARALLEL BINARY ADDER

Parallel binary adders are used to add two binary numbers. For example, if we want to add two four-bit numbers, we need to construct a parallel four-bit binary adder as shown in Figure 6.24. Such an adder requires one half-adder (denoted by HA) and three full-adders (denoted by FA). The binary numbers being added are  $A_4 A_3 A_2 A_1$  and  $B_4 B_3 B_2 B_1$ , and the answer is :

$$\begin{array}{r} A_4 A_3 A_2 A_1 \\ + B_4 B_3 B_2 B_1 \\ \hline S_5 S_4 S_3 S_2 S_1 \end{array}$$

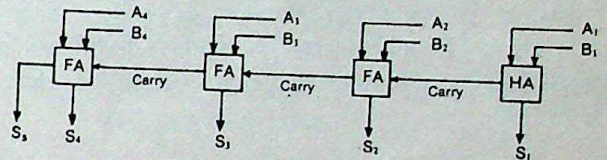


Figure 6.24. A parallel four-bit binary adder.

The first column requires only a half-adder. For any column above the first, there may be a carry from the preceding column. Therefore, we must use a full-adder for each column above the first.

To illustrate how the adder of Figure 6.24 works, let us see how it will add two numbers say 9 and 11. The binary equivalent of decimal 9 is 1001, and that of decimal 11 is 1011. Figure 6.25 shows the binary adder with these inputs.

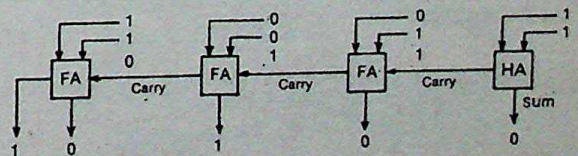


Figure 6.25. Example of adding two four-bit numbers using a parallel adder.



As shown in the figure, the half-adder adds 1+1 to give a sum of 0 and a carry 1. The carry goes into the first full-adder, which adds 0 + 1 + 1 to get a sum of 0 and a carry of 1. This carry goes into the next full-adder, which adds 0 + 0 + 1 to get a sum of 1 and a carry of 0. The last full-adder adds 1 + 1 + 0 to get a sum of 0 and a carry of 1. The final output of the system is 10100. The decimal equivalent of binary 10100 is 20 which is the correct decimal sum of 9 and 11.

The parallel binary adder of Figure 6.24 has limited capacity. The largest binary numbers that can be added using it are 1111 and 1111. So, its maximum capacity is :

$$\begin{array}{r} 15 \quad 1111 \\ + 15 \quad + 1111 \\ \hline 30 \quad 11110 \end{array}$$

In order to increase the capacity, more full-adders can be connected to the left end of the system. For instance, to add six bit numbers, two more full-adders must be connected and for adding eight bit numbers, four more full-adders must be connected to the left end of the full-adder of Figure 6.24.

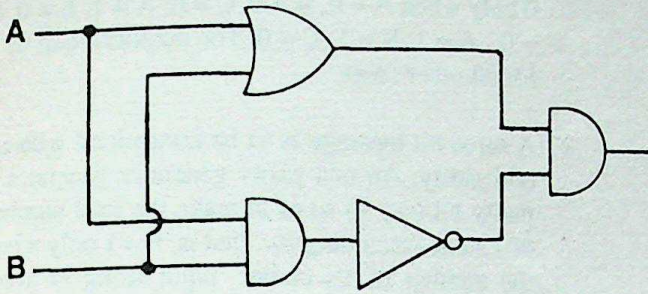
## QUESTIONS

1. Explain the principle of duality in Boolean algebra. How is it useful ?
2. Give the dual of the following Boolean expressions :
  - (a)  $\overline{A} + \overline{B}$
  - (b)  $A + \overline{B} + C$
  - (c)  $\overline{A.B} + A.\overline{B}$
  - (d)  $A+B$
  - (e)  $A.(A+B)$
  - (f)  $\overline{A} + A.B$
3. Give the dual of the rule  
 $A + \overline{A}.B = A + B$
4. Prepare a truth table for the following Boolean expressions :
  - (a)  $A.B + \overline{A}.B$
  - (b)  $A.\overline{B}.C + \overline{B}.\overline{C}$
  - (c)  $\overline{A} + \overline{B}$
  - (d)  $\overline{A} + \overline{B} + \overline{C}$
  - (e)  $A.B.\overline{C}$
  - (f)  $A.\overline{B}.C + \overline{A}.B.\overline{C}$
  - (g)  $(\overline{A} + \overline{B}).(\overline{A} + C).(\overline{B} + \overline{C})$
  - (h)  $\overline{A.C} + \overline{A.C}$

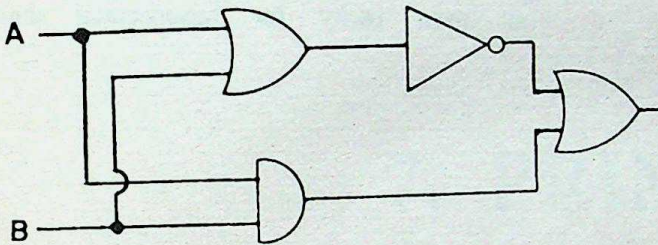
5. State and prove the two basic De Morgan's theorems.
6. Prove the following rules by the method of perfect induction.
  - (a)  $\overline{A.B} + A.B = A$
  - (b)  $A + \overline{A}.B = A + B$
  - (c)  $A . (A+C) = A$
  - (d)  $(A+B).(A.B) = A.\overline{B} + B.\overline{A}$
  - (e)  $(A+B).(A+B) = A.B + \overline{A}.\overline{B}$
7. Simplify the following Boolean expressions and draw logic circuit diagrams for your simplified expressions using AND, OR and NOT gates :
  - (a)  $\overline{x}.y.z + \overline{x}.y.\overline{z} + x.\overline{y}.\overline{z} + x.\overline{y}.z$
  - (b)  $\overline{x}.y.z + \overline{x}.\overline{y}.\overline{z} + x.y.z + x.y.\overline{z}$
  - (c)  $\overline{A}.C + \overline{A}.B + A.\overline{B}.C + B.C$
  - (d)  $\overline{A}.\overline{B}.\overline{C} + \overline{A}.B.\overline{C} + A.\overline{B}.\overline{C} + A.\overline{B}.C + A.B.\overline{C}$
  - (e)  $(A+B+C) . (A+\overline{B}+\overline{C}) . (A+B+\overline{C}) . (A+\overline{B}+C)$
  - (f)  $(A.B.C) . (A.B.\overline{C} + A.\overline{B}.C + \overline{A}.B.C)$
8. Find the complement of the following expressions :
  - (a)  $A.B + A.C$
  - (b)  $\overline{A}.\overline{B} + A.B$
  - (c)  $(A+B) . (B+C) . (A+C)$
  - (d)  $A . (B.C + \overline{B}.\overline{C})$
  - (e)  $A . (\overline{B} + \overline{C})$
  - (f)  $A . (B+C) . (\overline{C} + \overline{D})$
  - (g)  $A.B + (\overline{A}.\overline{B}) . (B.C + \overline{B}.\overline{C})$
9. Express the following Boolean functions in the sum-of-products form. Ensure that each term has all the literals.
  - (a)  $\overline{A}.(B+\overline{C})$
  - (b)  $(\overline{A}+B).(\overline{B}+\overline{C})$
  - (c)  $(A.B).(A.\overline{B}.\overline{C} + \overline{A}.C)$
  - (d)  $(\overline{A}+C).(\overline{A}+\overline{B}+\overline{C}).(A+\overline{B})$
  - (e)  $(A+B).\overline{C}$
  - (f)  $(\overline{A}+C).(A.B + A.C + B.C)$
10. Express the following Boolean functions in the product-of-sums form. Ensure that each term has all the literals.
  - (a)  $\overline{A} + B.\overline{C}$
  - (b)  $A.B + \overline{C}$
  - (c)  $A + B + C$
  - (d)  $(\overline{A}.\overline{B}).(\overline{A}.\overline{C} + \overline{B}.\overline{C})$
  - (e)  $(A.B).(\overline{B} + \overline{C})$
  - (f)  $A + A.B + \overline{A}.C$



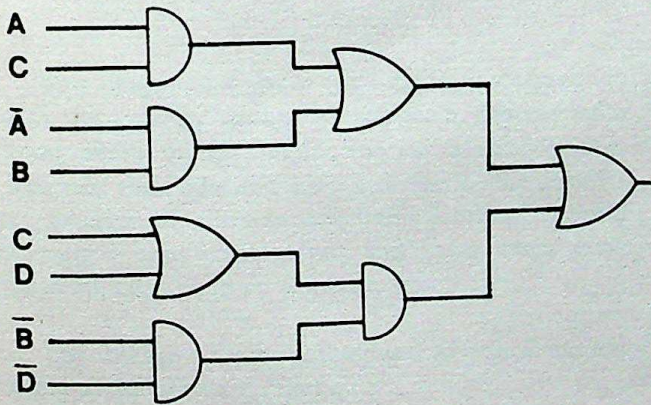
11. What will be the outputs of the following logic circuits for the specified inputs ?



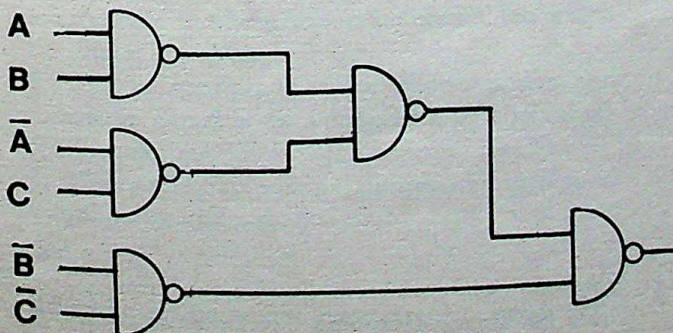
(a)



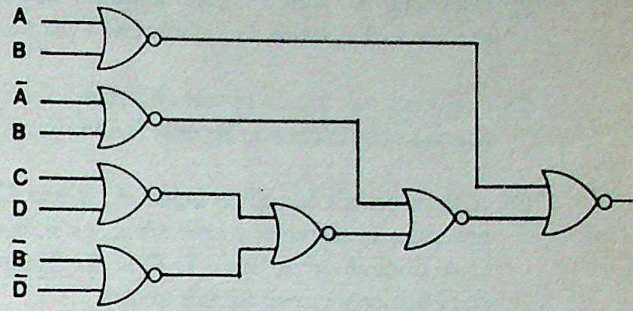
(b)



(c)



(d)



(e)

12. Construct logic circuit diagrams for the following Boolean expressions using AND/OR/NOT gates :

- $A \cdot B + A \cdot \bar{B}$
- $(A+B) \cdot (\bar{A} \cdot \bar{B})$
- $(\bar{A} + \bar{B}) \cdot (A+C) \cdot (B+\bar{C})$
- $A \cdot B + (\bar{A} \cdot \bar{B}) \cdot (B \cdot C + \bar{B} \cdot \bar{C})$
- $(A+B) \cdot (A+C) \cdot (\bar{A} + \bar{B})$

13. "AND, OR and NOT gates are logically complete." Discuss.

14. Why are NAND and NOR gates called universal gates ?

15. Show the implementation of the logical operations AND, OR and NOT only with NAND gates and only with NOR gates.

16. Construct logic circuit diagrams for the Boolean expressions of Question 12 using only NAND gates.

17. Construct logic circuit diagrams for the Boolean expressions of Question 12 using only NOR gates.

18. Construct logic circuit diagram for a half-adder using only NAND gates.

19. Construct logic circuit diagram for a half-adder using only NOR gates.

20. Why are combinational circuits more frequently constructed with NAND or NOR gates than with AND, OR and NOT gates ?

21. Prove that

$$(a) (A \oplus B) \oplus C = A \oplus (B \oplus C)$$

$$(b) (A \oplus B) \oplus C = A \oplus (B \oplus C)$$



22. Construct a logic circuit diagram for the exclusive-or function using only NOR gates.
23. Construct a logic circuit diagram for the equivalence function using only NAND gates.
24. A logic circuit has three inputs A, B and C. It generates an output of 1 only when  $A = 0, B = 1, C = 0$  or  $A = 1, B = 1, C = 0$ . Design a combinational circuit for this system.
25. A logic circuit has three inputs A, B and C. It generates an output of 1 only under the following conditions:  
 $A = 0, B = 0, C = 0$   
 $A = 0, B = 1, C = 1$   
 $A = 1, B = 0, C = 1$   
 $A = 1, B = 1, C = 1$

Design a combinational circuit for this system.

26. Design a gating network which will have output 0 only when  $A = 0, B = 0, C = 0$ ;  $A = 1, B = 0, C = 0$ ;  $A = 1, B = 1, C = 0$ . The outputs are to be 1 for all other cases.
27. A three bit message is to be transmitted with an odd parity. An odd parity generator generates a parity bit (say P) so as to make the total number of 1's odd (including P). That is,  $P = 1$  only when the number of 1's in the input string is even. Design a combinational logic circuit for such a parity generator.
28. Design a combinational logic circuit to generate an even parity for hexadecimal digits.



## CHAPTER 7

# 7. PRIMARY STORAGE

This chapter introduces the basic fundamentals related to the primary storage or the main memory of a computer system. In this chapter, you will learn about storage locations and addresses, how to determine the capacity of storage units, the difference between fixed and variable word-length storage organizations, and several other terms related to the main memory of a computer system.

Any storage unit of a computer system is ranked according to the following criteria :

1. *Access time.* This is the time required to locate and retrieve stored data from the storage unit in response to a program instruction. A fast access time is preferred.
2. *Storage capacity.* It is the amount of data that can be stored in the storage unit. A large capacity is desired.
3. *Cost per bit of storage.* An obvious goal is to minimize this cost.

Based on the above mentioned criteria, storage units are basically of two types - primary and secondary. As

compared to secondary storage units, primary storage units have faster access time, smaller storage capacity, and higher cost per bit of storage. In this chapter, we will be concentrating only on the concepts of primary storage. We shall learn about different types of secondary storage devices in the next chapter.

### STORAGE LOCATIONS AND ADDRESSES

A primary or internal storage section is basic to all computers. It is made up of several small storage areas called locations or cells. Each of these locations can store a fixed number of bits called *word length* of that particular primary storage. Thus, as shown in Figure 7.1, a given memory is divided into N words, where N generally is some power of 2. Each word or location has a built-in and unique number assigned to it. This number is called the *address* of the location and is used to identify the location. Each location can hold either a data item or an instruction, and its address remains the same regardless of its contents. The addresses normally start at 0 and the highest address equals the number of words that can be stored in the memory minus 1.



For example, if a memory has 1024 locations, then the address ranges between 0 and 1023. Thus, at address 0 we find a word, at address 1 a second word, at address 2 a third word, and so on upto the final word at the largest address.

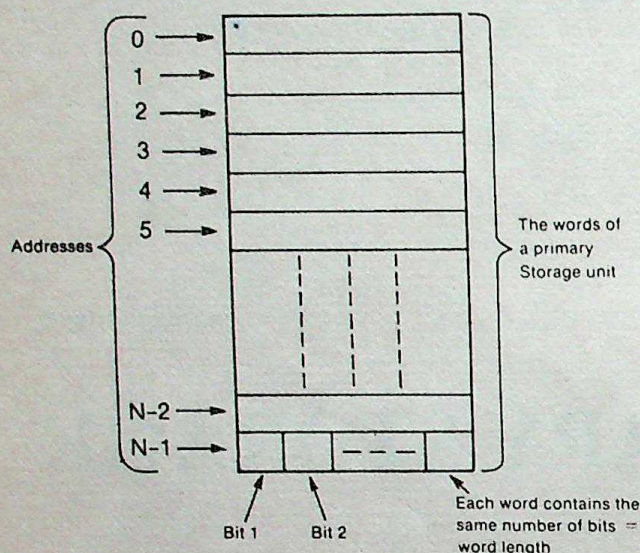


Figure 7.1. Organization of a Primary Storage Unit having N words.

There is an important difference between the address number and the contents of the address. A memory is like a large cabinet containing as many drawers as there are addresses in memory. Each drawer contains a word and the address of each word is written on the outside of the drawer. If we write or store a word say 10101010 at address 125, it is like placing the word 10101010 in the drawer labelled 125. Later, reading from address 125 is like looking in that drawer to see its contents which is now 10101010. We do not remove the word at an address when we read, but change the contents at an address only when we store or write a new word. Thus, entering data into a storage location is *destructive* of previous contents, but retrieving data from a location is *non-destructive*.

## STORAGE CAPACITY

The storage capacity of large computer systems is normally more than small systems. This capacity is defined in terms of bytes or words. A symbol commonly used to

denote the storage capacity of a computer is the letter K (Kilo) which is equal to  $2^{10}$ , or 1024. Thus a 32 Kilo-byte memory is capable of storing  $32 \times 1024 = 32,768$  bytes or characters. Memory sizes range from a few K bytes or words in small machines to several thousand K bytes or words in large machines. However, if the memory capacity is stated in terms of words, it is necessary to know the word size in bits or bytes in order to determine the actual storage capacity of the computer. Thus, while specifying the memory capacity in terms of words, it is customary to specify the total number of bits per word along with the total number of words. So a memory with 4096 locations each with a different address and with each location storing 16 bits is called a "16-bit 4096-word memory", or, in the vernacular of the computer trade, a "4 K 16-bit memory". Similarly, a memory having  $2^{15}$  words with each word of 16 bits is called a "32 K 16-bit memory". If the word size of a memory is 8 bits (equal to a byte) then it becomes immaterial whether the memory capacity is expressed in terms of bytes or words. Thus a memory having  $2^{16}$  words with each word of 8 bits is simply referred to as 64 K memory (word size of 8-bits is implicit here).

## WHY MORE BITS

You might have heard about 8-bit computers, 16-bit computers, 32-bit computers, etc. This refers to the word size of a particular computer in terms of total number of bits. Word size is an important architectural factor. Small machines have word sizes of 8 or 16 bits; large machines have word sizes of 32 bits or more. The obvious question that arises to one's mind is that what is the advantage of having more number of bits per word instead of having more words of smaller size?

For an answer to the above question, imagine a highway with eight lanes, and a heavy flow of traffic. If it is expanded to sixteen lanes, the flow of traffic speeds up considerably. '8-bits' refers to the number of 'lanes' on a microchip. Greater bits means a more rapid flow of electronic signals. In other words, a faster computer. Thus what an 8-bit computer takes one minute to do, a 32-bit computer can do in one second.

Data and instructions are moved to and from memory in word-sized bunches. Therefore, even if the electronic circuits used are comparable in speed, small machines are slower than big machines. This difference is analogous to providing the user of a small machine with a small shovel and the user of a large machine with a large shovel. Even though they both may be shoveling at comparable speeds, the user with the smaller shovel will be slower because more shovelfuls are needed to move the same amount of data.



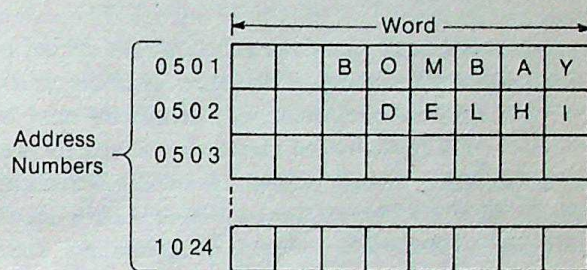
## FIXED AND VARIABLE WORD LENGTH STORAGE

The primary storage section of some computers is designed to store a fixed number of characters (equal to its word-length in bytes) in each numbered address location. Such computers are said to be *word addressable*, and they employ a *fixed word-length storage* approach. In these computers, storage space is always allocated in multiples of word-length. So if a word addressable computer has a fixed word-length of 4 characters then this computer will require one word (4 bytes) to store the word "CAT" and two words (8 bytes) to store the word "BOMBAY".

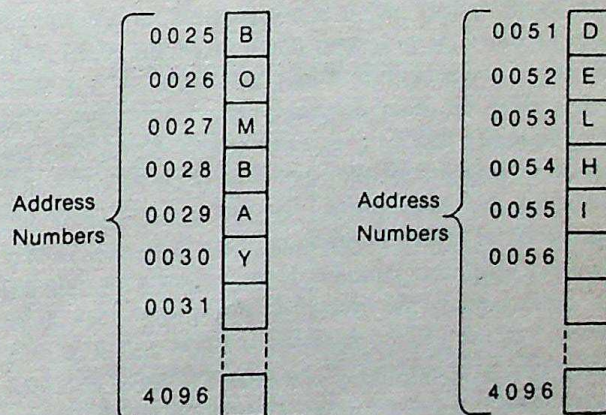
In many computers, the primary storage section is also designed in such a way that each numbered address can only store a single character (A, B, 1, 2, +, -, etc). Computers designed in this manner are said to be *character addressable* and they employ a *variable word-length storage* approach. Thus in these machines, only 3 bytes will be required to store the word "CAT" and only 6 bytes will be required to store the word "BOMBAY". Figure 7.2 summarizes the difference between the fixed-length and variable-length storage approach.

Both the fixed and the variable word-length storage systems have their own merits and demerits. The fixed word-length storage approach is normally used in large scientific computers for gaining speed of calculation. On the other hand, the variable word-length approach is used in small business computers for optimizing the use of storage space. For example, let us consider a fixed word-length machine with a word size of eight characters. If most of the data words to be stored are of less than five characters then more than half of the storage space will remain unused. This will not happen in case of a machine with variable word-length primary storage because a character can be placed in every storage cell of this machine. However, word addressable computers possess faster calculating capability because they can add two data words in a single operation. If the fixed-length word is eight characters, two eight digit numbers can be added in single operation. On the other hand, with a character addressable machine only one digit in each number is added during a single operation and eight steps would thus be needed to add two eight digit numbers.

Most of the today's business and scientific processing is handled by flexible computers which can employ either a fixed word-length or a variable word-length storage organization. The set of instructions available with these computers allow them to be operated as either variable or fixed word-length computers.



(a) Fixed word-length storage



(b) Variable word-length storage

Figure 7.2. Fixed word-length storage compared with variable word-length storage.

## RAM, ROM, PROM and EPROM

Primary storage is usually referred to as random access memory (RAM) because it is possible to randomly select and use any location of this memory to directly store and retrieve data and instructions. Each separate location inside the memory is as easy to access as any other



location, and takes the same amount of time. It is also referred to as read/write memory because information can be 'read' from a RAM chip and can also be 'written' into it.

A read only memory (ROM) is one in which information is permanently stored. The information from the memory can only be read and it is not possible to write fresh information into it. This is the reason why it is called ROM. When the power supply is switched off, the information stored inside a ROM is not lost as it is in the case of a RAM chip. Such memories are also known as field stores, permanent stores or dead stores.

The most basic computer functions are carried out by wired electronic circuits. However, there are several higher-level operations that are very frequently used but will require very complicated electronic circuits for their implementations. Hence instead of building electronic circuits for these operations, special programs are written to perform these operations. These programs are called *microprograms* because they deal with low-level machine functions and are essentially substitutes for additional hardware. Microprograms are written to aid the control unit in directing all the operations of the computer system. ROMs are mainly used by computer manufacturer's for storing these microprograms so that they cannot be modified by the users.

A variation of ROM chip is programmable read only memory (PROM). ROM chips are supplied by the computer system manufacturer and it is not possible for a customer (user) to modify the programs stored inside the ROM chip. However, it is possible for a user to "customize" a system by converting his own programs to microprograms and storing them in a PROM chip. Once the users programs are stored in a PROM chip, they can usually be executed in a fraction of the time previously required. PROMs are programmed to record information using a special facility known as a prom-programmer. However, once the chip has been programmed, the recorded information cannot be changed, i.e. the PROM becomes a ROM and it is only possible to read the stored information. PROM is also non-volatile storage, i.e. the stored information remains intact even if power is switched off.

Once information is stored in a ROM chip or a PROM chip it cannot be changed or altered. However, there is another type of memory chip called erasable programmable read only memory (EPROM) that overcomes this problem. As the name implies, it is possible to erase information stored in an EPROM chip and the chip can be reprogrammed to store new information using a special prom-programmer facility. Information stored in an EPROM chip is erased by exposing the chip for some time

to ultraviolet light. When an EPROM is in use, information can only be 'read' and the information remains on the chip until it is erased. EPROMs are mainly used by R & D personnel (experimenters) because they frequently change the microprograms to test the efficiency of the computer system with new programs.

Regardless of the type of ROM chip used, they all serve to increase the efficiency of a CPU by controlling the performance of a few specialized tasks. A generalized CPU can be made to meet the unique needs of different users merely by changing microprograms.

## CACHE MEMORY

A special very high speed memory is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. As you may be aware, CPU speeds are quite high compared to the access time of main memory. In many situations the performance of the processors are limited due to the slow speed of the main memory. A technique used to compensate for the mismatch in operating speeds is to employ an extremely fast, small memory between CPU and main memory whose access time is close to the processing speed of the CPU. This type of memory is called a *high-speed buffer* or *cache memory*. It is used to store segments of programs currently being executed and/or temporary data frequently needed in the present calculations. By making active programs and data available at a rapid rate, it is possible to increase the performance rate of the CPU.

As the name implies, cache memory is a memory in hiding and is not addressable by the user of the computer system. Its purpose is to look ahead and to provide the CPU with currently needed information. Cache memory makes main memory appear to be faster and larger than it really is. It is very expensive as compared to the main memory and hence its size is normally very small. The most important effect of cache memory is in improving the memory transfer rates and thus raising the processor speed.

## REGISTERS

As the instructions are interpreted and executed by the CPU, there is a movement of information between the various units of the computer system. In order to handle this process satisfactorily and to speed up the rate of information transfer, the computer uses a number of special memory units called *registers*. These registers are not considered as a part of the main memory and are used to retain information on a temporary basis.



The number of registers varies among computers as does the data-flow pattern. Most computers use several types of registers, each designed to perform a specific function. Each of these registers possess the ability to receive information, to hold it temporarily, and to pass it on as directed by the control unit. The length of a register equals the number of bits it can store. Thus a register that can store 8 bits is normally referred to as 8-bit register. Although the number of registers varies from computer to computer, there are some registers that are common to all computers. The function of these registers is described below.

1. *Memory Address Register (MAR)* : It holds the address of the active memory location. It is loaded from the program control register when an instruction is read from memory.
2. *Memory Buffer Register (MBR)* : It holds the contents of the memory word read from, or written in, memory. An instruction word placed in this register is transferred to the instruction register. A data word placed in this register is accessible for operation with the accumulator register or for transfer to the I/O register. A word to be stored in a memory location must first be transferred to the MBR, from where it is written in memory.
3. *Program Control Register (PC)* : It holds the address of the next instruction to be executed. This register goes through a step-by-step counting sequence and causes the computer to read successive instructions previously stored in memory. It is assumed that instruction words are stored in consecutive memory locations and read and executed in sequence unless a branch instruction is encountered. A *branch instruction* is an operation that calls for a transfer to a non-consecutive instruction. The address part of a branch instruction is transferred to the PC register to become the address of the next instruction. To read an instruction, the contents of the PC register are transferred to the MAR and a memory read cycle is initiated. The instruction placed in the MBR is then transferred to the instruction register.
4. *Accumulator Register (A)* : This register holds the initial data to be operated upon, the intermediate results, and also the final results of processing operations. It is used during the execution of most instructions. The results of arithmetic operations are returned to the

accumulator register for transfer to main storage through the memory buffer register. In many computers there are more than one accumulator registers.

5. *Instruction Register (I)* : It holds the current instruction that is being executed. As soon as the instruction is stored in this register, the operation part and the address part of the instruction (see Chapter 12) are separated. The address part of the instruction is sent to the MAR while its operation part is sent to the control section where it is decoded and interpreted and ultimately command signals are generated to carry out the task specified by the instruction.
6. *Input/Output Register (I/O)* : This register is used to communicate with the input/output devices. All input information such as instructions and data is transferred to this register by an input device. Similarly, all output information to be transferred to an output device is found in this register.

Table 7.1 summarises the functions of each of these registers.

Table 7.1. Functions of Various Registers

| SE No. | NAME OF REGISTER     | FUNCTION                                                 |
|--------|----------------------|----------------------------------------------------------|
| 1.     | Memory Address (MAR) | Holds the address of the active memory location          |
| 2.     | Memory Buffer (MBR)  | Holds information on its way to and from memory.         |
| 3.     | Program Control (PC) | Holds the address of the next instruction to be executed |
| 4.     | Accumulator (A)      | Accumulates results and data to be operated upon         |
| 5      | Instruction (I)      | Holds an instruction while it is being executed          |
| 6.     | Input/Output (I/O)   | communicates with the I/o devices                        |



## QUESTIONS

1. Differentiate between a bit, a byte, and a word.
2. What is a memory address? Describe with an example.
3. Explain the difference between an address and the contents of an address.
4. Write a note on the 'storage capacity' of a computer.
5. How many bytes can be stored in the primary storage unit of a 512 K-byte computer?
6. What is the memory capacity in terms of bytes of a 128 K 16-bit computer?
7. What is the importance of the difference in word sizes between small and large computers?
8. Distinguish between word-addressable and character-addressable computers. Discuss their relative advantages and disadvantages.
9. How many bytes will be required to store the word 'MEMORY' in (a) a character-addressable computer (b) a word-addressable computer having word-length of 64 bits?
10. What is a microprogram?
11. How is read only memory used?
12. Differentiate between RAM, ROM, PROM and EPROM.
13. What is a cache memory? How is it different from a primary memory?
14. What are registers? Name some of the commonly used registers and briefly describe the function of each.
15. Give the full form of the following abbreviations:

|          |           |
|----------|-----------|
| (a) K    | (e) EPROM |
| (b) RAM  | (f) MAR   |
| (c) ROM  | (g) MBR   |
| (d) PROM |           |



## CHAPTER 8

# 8. SECONDARY STORAGE DEVICES

It is desirable that the operating speed of the primary storage of a computer be as fast as possible because most of the transfers of data to and from the processing unit is via the main memory. For this reason, storage devices with very fast access times, such as semiconductors, are generally chosen for the design of main memory. These high-speed storage devices are expensive and hence the cost per bit of storage is also high for a primary storage. Thus the storage capacity of the main memory of a computer system is limited. Often, it is necessary to store many millions, sometimes billions, of bytes of data. Unfortunately, the storage capacity of the primary storage of today's computers is not sufficient to store the large volume of data handled by most data processing centres. As a result, additional memory, called the *auxiliary memory* or *secondary storage*, is used with most computer systems. This section of the computer's memory is characterised by low cost per bit stored, but it generally has an operating speed far slower than that of the primary storage. This section of the memory is also referred to as *backup storage* because it is used to store large volume of data on a permanent basis which can be partially transferred to the primary storage as and when required for processing. Data are stored in secondary storage in the same binary codes as

in main storage and are made available to main storage as needed. A wide range of secondary storage devices is available. Typical hardware devices are magnetic tape and magnetic disk. In this chapter you will learn about many different types of secondary-storage devices.

## SEQUENTIAL AND DIRECT-ACCESS DEVICES

Several different devices can be used as a secondary storage, but the one selected for a particular application mainly depends upon how the stored information needs to be accessed. Basically there are two methods of accessing information - sequential or serial access and direct or random access. A *sequential-access device* is one in which the arrival at the location desired may be preceded by sequencing through other locations, so that access time varies according to location. In other words, information on a serial device can only be retrieved in the same sequence in which it is stored. Sequential processing is quite suitable for such applications like preparation of monthly payslips, or monthly electricity bills, etc., where each address needs to be accessed in turn. In these applications, data for each and every employee or customer needs to be processed at



scheduled intervals (in this case monthly). However, while working with a sequential-access device, if an address is required out of order, it can only be reached by searching through all those addresses which are stored before it. For instance, data stored at last few locations cannot be accessed until all preceding locations in the sequence have been traversed. Magnetic tape and punched paper media are typical sequential-access storage devices.

In many applications, we need to access information in a more direct manner than serial devices allow. For example, in a computerised bank, at any instant it is required to determine the exact balance in the savings account of a particular customer. Similarly, in a computerised airline ticket booking system, immediate access may be required to reservation system records to find out if seats are currently available on a particular flight. In such applications, if we use sequential-access device, the time taken to access the desired information may be enormous which will cause frustration to the customer. Backing storage devices exist which permit access to individual information in a more direct or immediate manner. These direct devices are also called random-access devices because the information is literally available at-random, i.e., it is available in any order. Thus a *random-access storage device* is one in which any location in the device may be selected at random, access to the information stored is direct, and approximately equal access time is required for each location. Magnetic disk and magnetic drum are typical direct-access storage devices. We will now discuss in detail some of the sequential and direct-access secondary storage devices.

### PUNCHED PAPER TAPE

It is a sequential-access secondary storage device. Data is coded on paper tape in the form of punched hole combinations. The tape is normally 1 inch wide, comes in rolls, and may be used in any length upto several hundred feet. There is a line of sprocket holes in the middle of the tape for the purpose of feeding the tape through the tape punching and reading device. Information is recorded as holes punched in rows across the width of the tape, with one row representing one character (Figure 8.1). The maximum number of holes per row is referred to as the number of *channels* on the tape. The particular coding system used to record information on the tape depends on the number of channels. An eight channel tape is the one most commonly used.

The number of characters that can be stored in a given physical space is low for paper tape storage. Moreover, the paper tape media is easily torn and mutilated. Hence, the use of paper tapes as a secondary

storage device is very rare today and is vanishing day by day. However, the storage capacity of punched paper tape is virtually unlimited, and the cost per bit stored is very low.

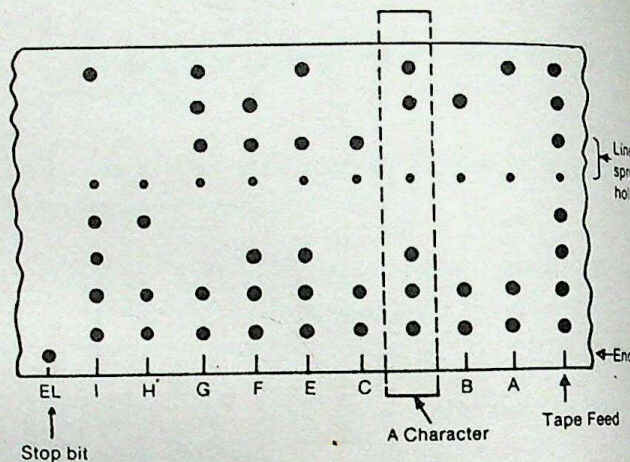


Figure 8.1. An eight channel punched paper tape

### MAGNETIC TAPE

Magnetic tape is one of the most popular storage medium for large data that are sequentially accessed and processed. The tape is a plastic ribbon usually 1/2 inch wide that is coated on one side with an iron-oxide material which can be magnetized. The tape ribbon itself is stored in rolls of 50 to 2400 feet or a small *cartridge* or *cassette*, similar to the tape used on a tape recorder except that it is of higher quality and more durable. Like recorder tape, computer tape can be erased and reused indefinitely. Data on a tape are automatically erased as new data is recorded in the same area.

Information is recorded on the tape in the form of tiny invisible magnetized and non-magnetized spots (representing 1's and 0's) on the iron-oxide side of the tape. The tape is divided into vertical columns called *frames* and horizontal rows called *channels* or *tracks*. A character is recorded per frame using one of the computer codes described in Chapter 4. As shown in Figure 8.2, older tapes had 7 tracks and they used the 6-bit BCD code format for data recording. As in BCD code, the letter A is represented on this tape by the code 110001. The first six tracks are used for recording the 6 bits of the BCD code. The seventh track is used for recording the parity bit.



A *parity bit* or *check bit* is used to detect errors that occur due to the loss of a bit from a string of 6 or 8 bits during data input or output operations. If the basic code for a character requires an odd number of 1 bits (such as 1, 2, or 3 in BCD) an additional 1 bit is added to the check bit position so that there will always be an even number of 1 bits. This is an example of *even-parity*. Similarly, in *odd-parity*, the check bit is used to always produce an odd number of 1 bits. That is, the check bit will be 1 if the total number of 1 bits for representing a particular character is even and it will be 0 otherwise. The tape shown in Figure 8.2 uses the parity bit for an even parity.

Most of the modern magnetic tapes have 9 tracks they use the 8-bit EBCDIC code format for data recording. As shown in Figure 8.3, in this case the fourth track is used for parity bit which produces an odd parity in each character. Mark that the letter A is represented here by the code 11000001 as in the 8-bit EBCDIC format.

*Magnetic tape drive* is a machine that can either read from a tape into the CPU or it can write the information being produced by the computer onto a tape. As you can see in Figure 8.5 that the tape on a reel moves through a tape drive in much the same way that a film moves through a movie projector. During processing, the tape moves from a supply reel to a take-up reel via two transport channels and through a read/write head assembly. The read/write head assembly is a single unit having one read/write head for each tape track. They either read information, or write information on the tape. The two transport channels are designed to take up slack tape, acting as buffers to prevent the tapes from snapping or stretching when starting from a stationary position or slowing down to full speed. Several methods are used to prevent tape damage from sudden bursts of speed and different tape drives may use different mechanisms for this purpose.

*Data transfer rate* or data rate is one measure of the capacity of a secondary storage device. The data transfer rate of a magnetic tape is the product of tape density and tape speed. *Tape density* is the number of frames that can be compressed into 1 inch of tape. Because a frame contains a character, or a byte, tape densities are expressed in characters per inch (CPI) or bytes per inch (BPI). Typical tape densities are 556 BPI, 800 BPI, 1600 BPI, 3250 BPI or 6250 BPI, with the higher densities applicable to the modern systems. *Tape speed* is measured in inches per second, and high-performance tapes have speed in excess of 200 inches per second. Thus, for a magnetic tape, the data transfer rate can be in excess of 1 million bytes per second.

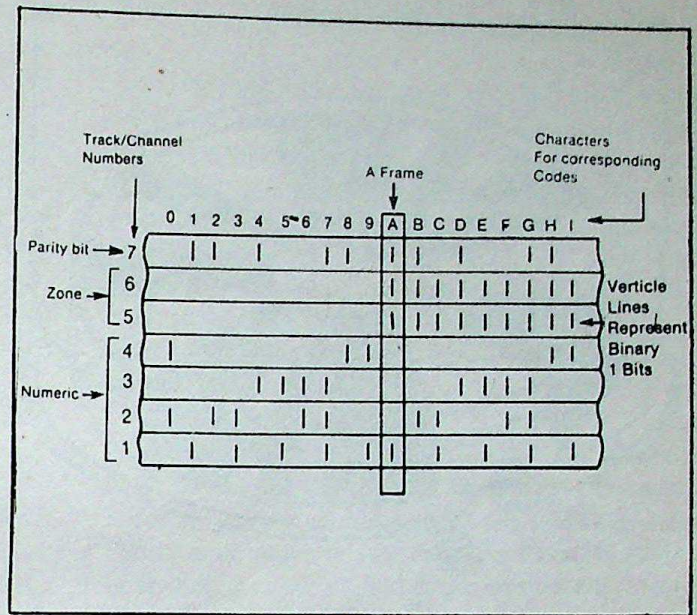


Figure 8.2. Data recording on a 7 track magnetic tape using 6-bit BCD code format. Here the 7th track is used for an even parity so all frames contain an even number of 1 bits.

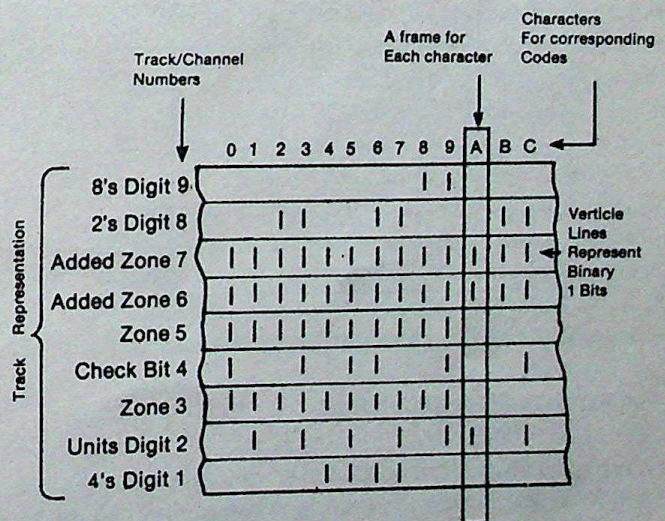


Figure 8.3. Data recording on a 9 track magnetic tape using 8-bit EBCDIC code format. This figure illustrates a so-called odd parity tape on which all the frames contain an odd number of 1 bits.



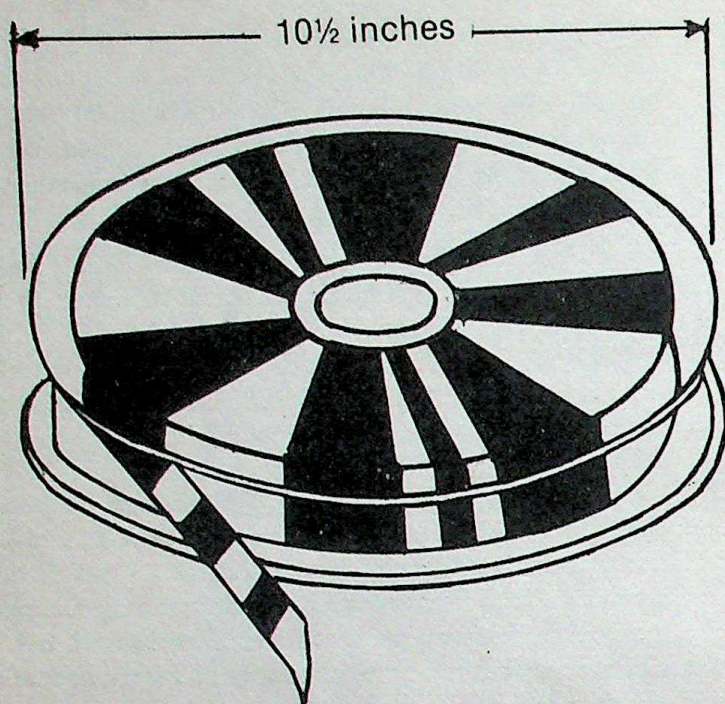


Figure 8.4. Magnetic tape in reel form.

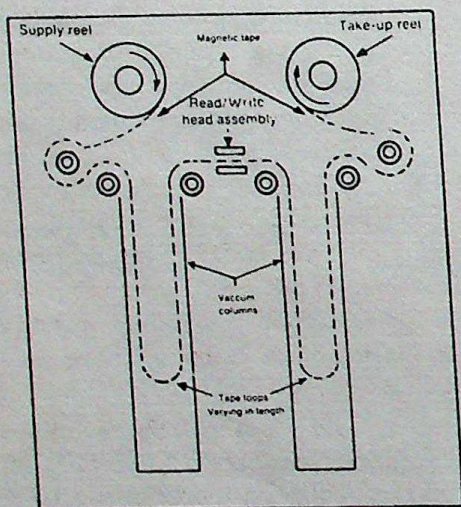


Figure 8.5. A magnetic tape drive.

### Advantages of Magnetic Tape

1. *Unlimited storage* : The storage capacity of a magnetic tape is virtually unlimited because we can use as many tapes as required for recording our data.

2. *High data density* : A typical 10.5 inch reel of magnetic tape is 2,400 feet long and is able to hold 800, 1600, or 6250 characters per inch of this length. Thus, if 6250 characters are held per inch and if the tape is 28,800 inches long (2400 feet times 12 inches), then the maximum capacity of the tape is 180 million characters.

3. *Low Cost* : A 10.5 inch reel of tape costs Rs400/- (about US \$32). This is much less as compared to other data storage devices. An additional cost benefit is that tapes can be erased and reused many times. Punched tapes do not have this advantage.

4. *Rapid transfer rate* : As already discussed, the transfer rate for a magnetic tape can be in excess of 1 million bytes per second.

5. *Ease of handling* : Since the reel is compact and weighs less than 3 pounds, it obviously takes up much less storage space and is much easier to handle. Large computer centres make use of extensive tape libraries, which contain thousands and thousands of reels of stored information and may occupy an entire room in an organization.

6. *Portability* : A reel of tape is also a convenient way of carrying information from one place to another. It is often used for transferring data and programs from one computer to another which are not linked together.

### Limitations of Magnetic Tape

1. *No direct access* : Magnetic tape is a sequential access device and hence data recorded on tape cannot be addressed directly. They can only be retrieved sequentially. If a data item is at the end of a tape, all the earlier data have to be read before accessing the required information. If random access is frequently required then magnetic tape is not a suitable storage media for such type of data. Much operator time would be required to load and unload tapes, and too much machine time would be wasted in fetching the exact data that is needed.

2. *Indirect interpretation* : Data stored on magnetic tape are in the form of tiny invisible magnetized and non-magnetized spots. Hence the contents of the tape cannot be interpreted and verified directly. An interpretation or printing run must be made if the accuracy of tape data is questioned. This calls for the need of an interpretation of the stored data.

3. *Environmental problems* : Specks of dust, uncontrolled humidity or temperature levels can cause reading errors. Moreover, tapes and reel containers must be stored in a dust free environment and should be properly labeled so that some useful data stored on a particular tape is not erased by mistake.

### TAPE CASSETTES AND CARTRIDGES

Magnetic tape reels are suitable for use on large and medium size computers. The same magnetic



is used in smaller systems in the form of cassettes and cartridges. The changeable tape cassette used in the familiar home recorder is now becoming an attractive means for recording digital data because they are small, changeable, and inexpensive. Unfortunately, the quality of tape and the tape moving mechanism in the conventional home tape cassette are not good enough for computer usage. As a result, a number of high-quality digital cassettes have been developed that are very similar in appearance to the home cassettes. Figure 8.6 shows a tape cassette.

There are also large tape cartridges which contain long strips of magnetic tape and which resemble large cassettes. These cartridges provide a more *convenient* way to package tape, and greatly simplify the mounting of tape which is normally a problem with conventional reels of tape that have to be manually mounted on the mechanism. Moreover, tape cartridges also provide protection against dirt and contamination, since the tape is sealed in the cartridge. A tape cartridge is shown in Figure 8.7. Tape cassettes and cartridges have relatively low data-transfer-rate which is typically less than 10,000 bytes per second.

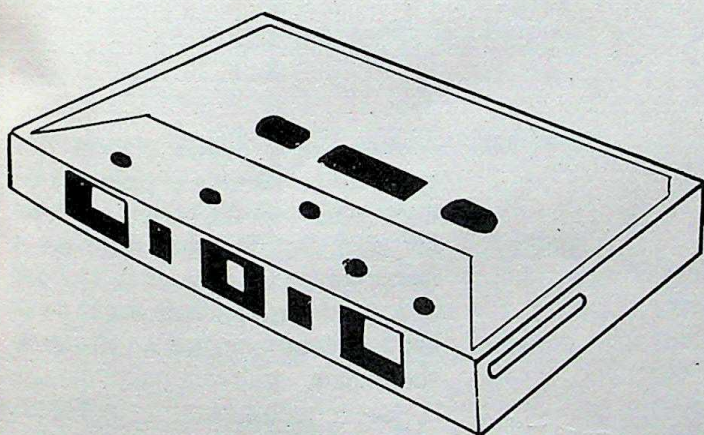


Figure 8.6. Magnetic tape cassette.

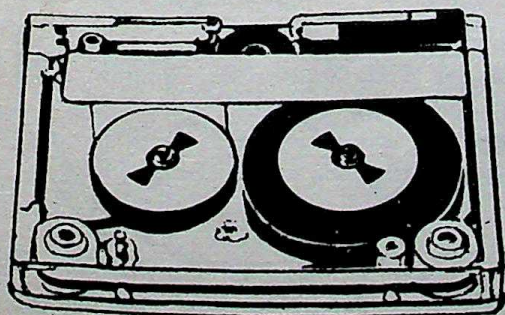


Figure 8.7. Magnetic tape cartridge.

## MAGNETIC DISK

A magnetic disk is a thin, circular metal plate/platter coated on both sides with a magnetic material. It is very similar in appearance to a LP gramophone record. A *disk pack* consists of a number of these disks, three or more, mounted about half-an-inch apart from each other on a central shaft which rotates at speeds of 2,400 or more revolutions per minute (rpm). Thus all the disks of a disk pack move simultaneously in the same direction and at equal speed. Magnetic disks are the most popular medium for direct-access secondary storage.

*Storage of Information.* In a disk pack, information is stored on both the surfaces of each disk plate except the upper surface of the top plate and the lower surface of the bottom plate which are not used. As shown in Figure 8.8, each disk consists of a number of invisible concentric circles called *tracks*. A set of corresponding tracks in all the surfaces is called a *cylinder* (Figure 8.9). Thus a disk pack having 10 disk plates will have 18 recording surfaces and hence it will have 18 tracks per cylinder. Each track is further subdivided into sectors (Figure 8.10).

Information is recorded on the tracks of a disk surface in the form of invisible tiny magnetic spots. The presence of a magnetized spot represents a 1 bit and its absence represents a 0 bit. A standard binary code, usually 8-bit EBCDIC, is used for recording data. In some systems, the outer tracks, contain more bits than the inner tracks, because the circumference of an outer track is greater than that of an inner track. However in most systems, each track contains the same number of characters, which means that the outer tracks of the disk are less densely packed with characters than those towards the centre.

The information stored on a disk can be read many times without affecting the stored data. So the reading operation is non-destructive. But the writing of new data erases data previously stored at that location of the disk. The data stored on a magnetic disk remains indefinitely until they are erased and reused at a future time.

*Storage Capacity.* The more disk surfaces a particular disk pack has, the greater will be its storage capacity. But the storage capacity of disk system also depends on the tracks per inch of surface and the bits per inch of track. Although the diameter of a standard sized disk is 14 inches, some disks are quite large running upto 4 feet in diameter. Larger disks have more tracks and hence they have greater storage capacity.

The total number of bytes that can be stored in a

disk pack is:



Number of cylinders x Tracks per cylinder x Sectors per track x Bytes per sector.

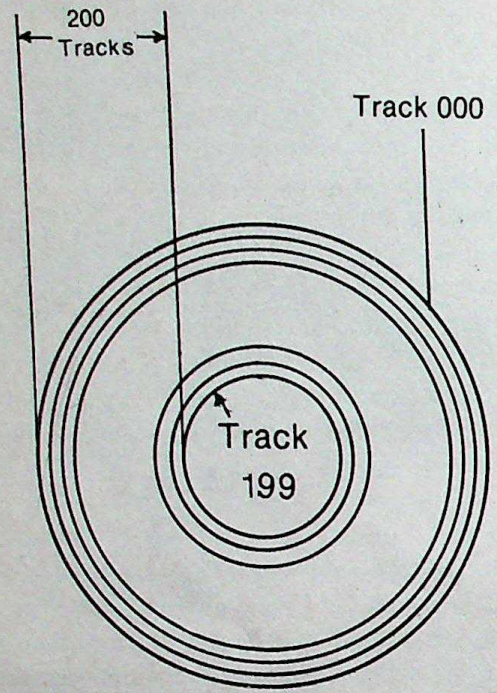


Figure 8.8. Tracks on a disk. The number of tracks varies (often 200 or more). The outer track is numbered 000 and inner track number is one less than total number of tracks. The inner track on a disk with 200 tracks is numbered 199.

For a typical example, let us assume that a disk pack has 10 disk plates each having 200 tracks. Suppose there are 40 sectors per track and each sector can store 256 bytes. Since the disk pack has 10 disk plates, so it will have 18 recording surfaces or in other words 18 tracks per cylinder. 200 tracks per plate means that there are 200 cylinders. So the capacity of this disk pack is :  $200 \times 18 \times 40 \times 256 = 36864000$  bytes or 36.864 million bytes.

Disk packs are potentially very high capacity storage devices typically in the range 20 to 1000 megabytes (M bytes). One megabyte is equal to  $10^6$  bytes. Essentially one character can be stored per byte. So a 50 M byte disk pack has the capacity to store 50 million characters of information.

**Accessing of Data.** Data are recorded on the tracks of a spinning disk surface and read from the surface by one or more *read/write heads*. There are two basic types of disk systems - moving-head and fixed-head. The *moving-head-system* consists of one read/write head for each disk surface mounted on an access arm which can be moved in and out (Figure 8.11). So in this system, each read/write head moves horizontally across the surface of the disk so that it is able to access each track individually. Each usable

surface of the disk pack has its own head and all the heads move together. Information stored on the tracks which

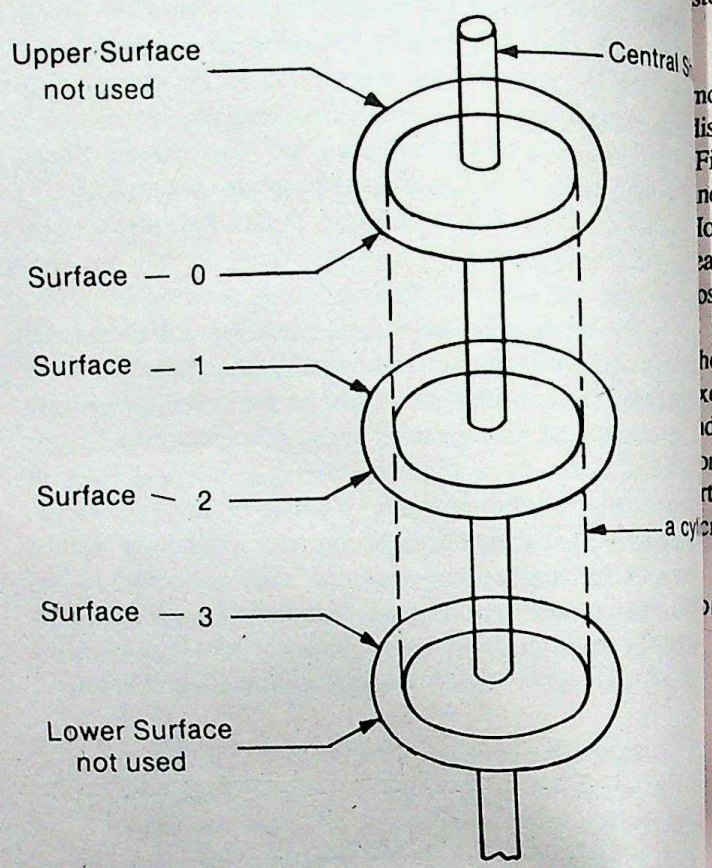


Figure 8.9. A disk having 3 disk platters. The upper surface of the top plate and the lower surface of the bottom plate are not used. So altogether there are 4 usable surfaces numbered 0, 1, 2 and 3. A set of corresponding tracks on the 4 surfaces is called a cylinder as shown here.

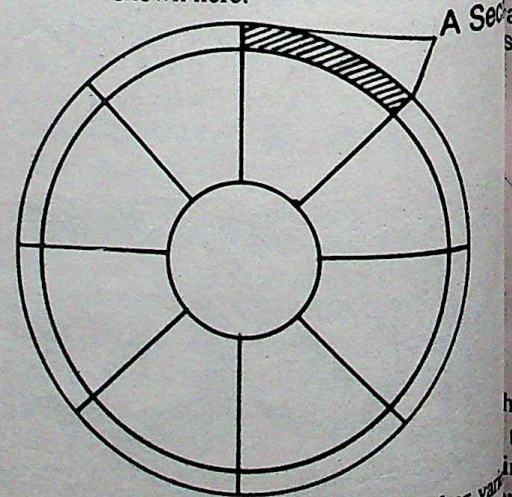


Figure 8.10. Sectors of a disk. The number varies but there are often 8 or more sectors per track.



constitute a cylindrical shape through the disk pack are therefore accessed simultaneously. Note the cylindrical storage arrangement of information in a disk pack.

In the *fixed-head system*, the access arm is non-movable. A large number of read/write heads are distributed over the disk surfaces, one head for each track (figure 8.12). As a result, no head movement is required and therefore information is accessed more quickly. However, because of the space required for the additional read/write heads, fixed-head disks have less capacity and cost more per byte of data stored than moving-head disks.

The disk pack on some disk storage devices is permanently fixed in position, while on others, the pack can be removed and replaced by another one in a matter of seconds. So the storage capacity of a removable disk pack system is virtually unlimited because the storage space can be increased without the heavy expense of buying

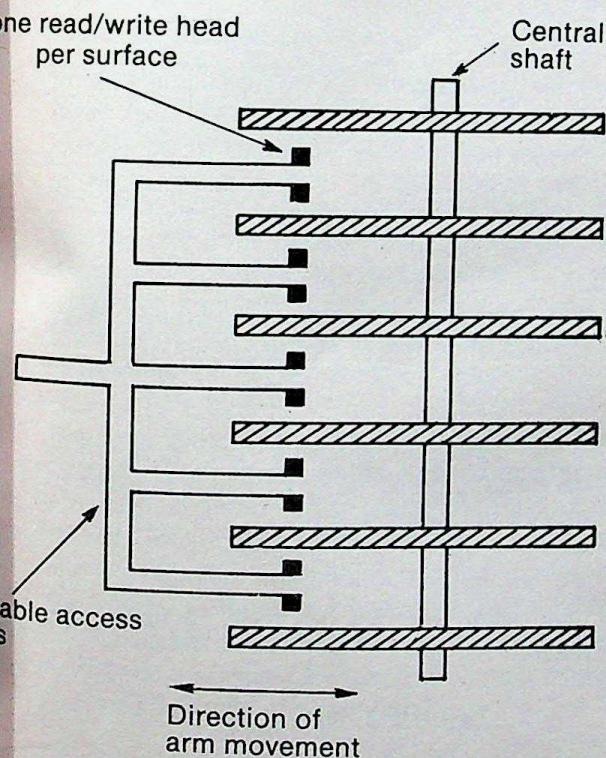


Figure 8.11. Vertical cross section of a moving-head disk system. There is one read/write head per surface. The heads move horizontally across the surface of the disk.

er complete device. In this context, it is important to that exchangeable disk packs are only associated with moving-head systems. The disk packs of a fixed-head system are non-removable.

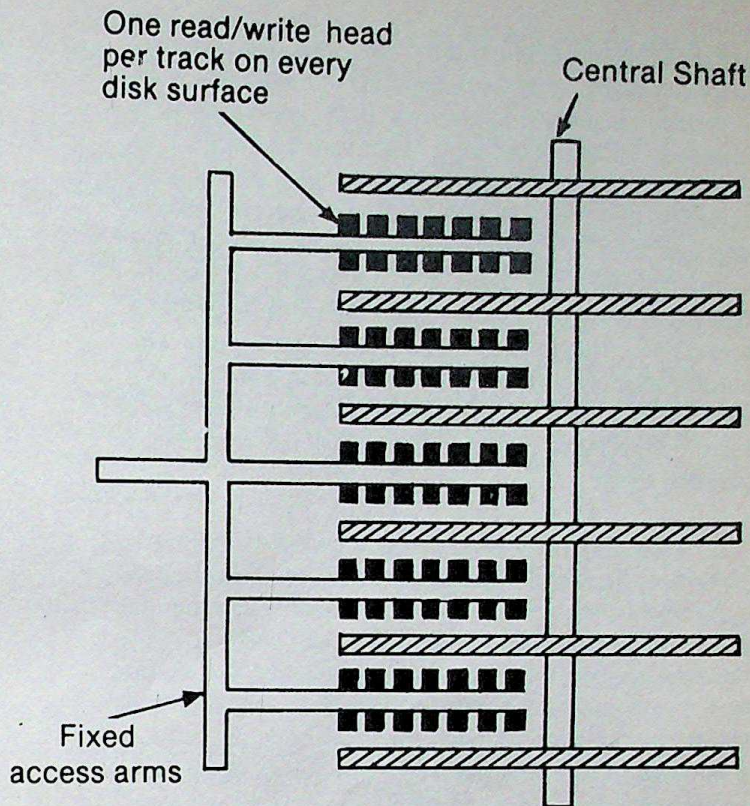


Figure 8.12. Vertical cross section of a fixed-head disk system. There is one read/write head per track on every disk surface. No head movement is required and so the access arms are non-movable.

Another point to be noted is that in case of both the fixed-head and the moving-head systems, the read/write heads are of flying type, i.e., they do not have direct contact with the surface of the disks. When a disk rotates at a high speed, a thin layer of air rotates with the disk. The head is so shaped that it rides on this layer of rotating air, thus maintaining a separation of about  $1/400$ th of an inch from the disk surface. This prevents wear on the surface of the disk.

**Access Time.** In order to access information from a disk, the disk address of the desired data has to be specified. The disk address is specified in terms of the track number, the surface number, and the sector number. Information is always written from the beginning of a sector and can be read only from the track beginning.

As soon as a read/write command is received by the disk unit, the read/write heads are first positioned on to the specified track number by moving the arm assembly in the proper direction. This involves a mechanical motion of the arms and is slow. The time required to position the head



over the proper track is called the *seek time*. The seek time varies depending on the position of the arm assembly when a read/write command is received. If the arm assembly is positioned on the outer most track and the track to be reached is the inner most one then the seek time will be maximum, and it will be zero if the arm assembly already happened to be on the desired track. The average seek time is thus specified for most systems which is generally somewhere between several milliseconds to fractions of a second. Note that seek time is associated only with movable-head systems. For a fixed-head system, the seek time is always zero because there is a head for each track and no movement of head is required for accessing a particular track.

Once the heads are positioned on the desired track, the head on the specified surface is activated. Since the disk is continuously rotating, this head should wait for the desired data (specified sector) to come under it. This rotational waiting time, i.e., the time required to spin the needed data under the head is called the *latency time*. The latency time is also a variable and depends on the distance of the desired data from the initial position of the head on the specified track. It also depends on the rotational speed of the disk. An average latency time is thus normally specified which is of the order of 10 to 15 milliseconds.

The total access time for a disk is equal to the seek time plus the latency time. The average access time for most disk systems is usually between 10 and 100 milliseconds. Technically speaking, disk systems have direct but not random access to the stored data. *Random access* refers to a storage device in which the access time is independent of the physical location of the data. For example, primary storage is a random access storage. Since the disk access time is dependent on the physical location of data, it is more correct to say that disks provide *direct access*. This distinction is not always observed and hence disk systems are sometimes referred to as random access storage devices.

The data transfer rate of a disk system depends on the density of the stored data and the rotational speed of the disk. Maximum rate usually ranges between 400,000 and 2 million characters per second.

### Comparison Between Magnetic Tape And Magnetic Disk Storage

The following are the relative advantages and disadvantages of a magnetic tape and a magnetic disk storage systems :

1. Magnetic disk has the flexibility of being used as a sequential as well as a direct access storage device. On the other hand, magnetic tapes can be used only for sequential processing of data.
2. Magnetic disks are less vulnerable to damage from dust or careless handling than magnetic tapes.
3. Any information desired from a disk storage can be accessed in a few milliseconds because it is a direct access storage device. This is not possible in case of a tape storage which is a sequential access storage device.
4. Data transfer rate for a magnetic disk system is normally higher than a tape system.
5. A disk pack equal in storage capacity to a reel of magnetic tape may be about 25 times more expensive. On a cost-per-bit basis, the cost of disks is low. But the cost of magnetic tape is even less.
6. Sequential processing using disks may be slower and less efficient than when tapes are used.
7. It is easier to maintain the security of information stored on a tape as compared to information stored on a disk.
8. Disk packs are not so easily portable as magnetic tapes.

### FLOPPY DISK

The metallic disks that we described above are called hard-disks. These disks, in the form of disk-packs, are suitable only for large and medium sized computers and are often too expensive for small computer systems. An increasingly popular direct access secondary storage medium for micro and mini computer systems is the flexible, or floppy, disk. Floppy disks are also referred to as diskettes or floppies. They were introduced by IBM in 1971 and are now being produced in various sizes by many manufacturers.



A floppy disk is made of flexible plastic which is coated with magnetic oxide. The flexible disk is enclosed within a square plastic or cardboard jacket, often referred to as a cartridge. The jacket gives handling protection to the disk surface. Moreover, it has a special liner that provides a wiping action to remove dust particles that are harmful for the disk surface and the read/write head. The disks are mounted on the disk drive along with the jacket cover and information is written and read through an aperture in the jacket (Figure 8.13). The cartridge can be very easily loaded into, and unloaded from, a drive unit. Unlike the hard-disk drives, the read/write heads of a floppy-disk unit make direct contact with the disk surface during the process of reading or writing, and floppy disks therefore get worn with constant use.

Floppy disks are typically 3, 5.25, or 8 inches in diameter. They come in either single - or double - density versions and record on one or both surfaces of a diskette. Thus there are basically four types of diskettes:

- (a) single-sided-single-density,
- (b) single-sided-double-density,
- (c) double-sided-single-density, and
- (d) double-sided-double-density.

The capacity of diskettes varies depending on their size (diameter) and type (mode of data recording). Thus, the capacity of a single-sided-single-density 5.25 inch diskette may only be a little over 100 kilobytes (1 kilobyte  $10^3$  bytes), while the capacity of a double-sided-double-density 8 inch diskette may be almost 2 megabytes.

Floppy disks are very cheap as compared to other storage devices. They greatly enhance the on-line storage capacity of small systems at affordable price. They are also convenient off-line storage medium for the small system users. *On-line* means connected to a computer system and under the control of the central processing unit. *Off-line* means not connected to a computer system. Diskettes are currently the most popular, inexpensive storage medium used in microcomputers.

## WINCHESTER DISK

There is a third type of relatively new disk storage known as winchester disk. In this unit disks are permanently housed in sealed, contamination-free containers. The disks are coated with a special lubricant which reduces the friction when the read/write heads land on the disk surface. The container is usually not removed from the disk drive. High-capacity systems using these sealed housings are said to employ winchester technology. This technology enables greater precision of alignment, an

increase in the number of tracks on the disk surface and a higher storage density per track.

Winchester disks are fast and highly reliable, yet low priced compared with conventional hard-disk devices. They are normally 5.25, 8, or 14 inches in diameter and storage capacities of 10, 20, and 40 megabytes are typical. Some winchester units employ dual disk drives in which case the storage capacity is doubled.

Winchester disks are used in all but the smallest computer systems. They are extensively used to support minicomputers and their smaller versions are now competing with floppies to make their place in microcomputer systems.

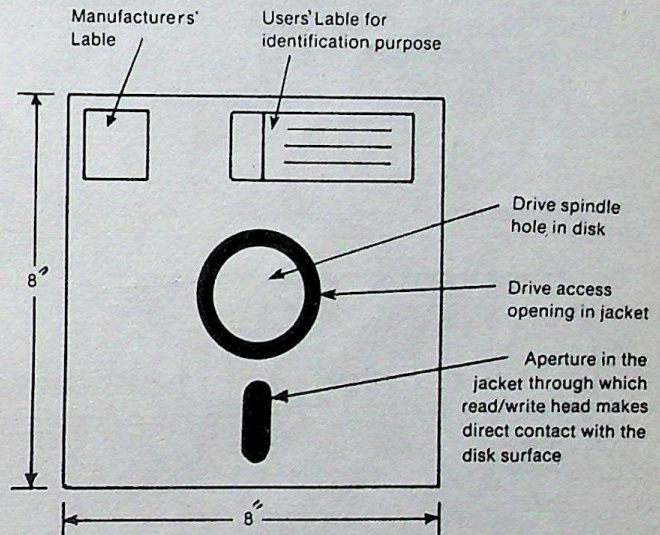


Figure 8.13. An 8 inch floppy disk enclosed within jacket. The drive mechanism clamps onto a portion of the disk exposed by the drive access opening in the jacket.

## MAGNETIC DRUM

Like the magnetic disk, magnetic drum is a direct access storage device that can be used for both sequential and random processing. It consists basically of a cylinder whose outer surface is coated with a thin layer of magnetizable material. A motor rotates the cylinder on its axis at a constant and rapid rate. As shown in Figure 8.14,



the surface of the drum is divided into tracks upon which data is stored as magnetized spots in the same manner as on the surface of a disk. Data is recorded on the rotating drum and read from the drum by a set of stationary read/write heads, which are positioned a fraction of an inch from the drum surface. The writing of new data on the drum erases data previously stored at that location. However, the data recorded on the drum surface will remain indefinitely until they are erased.

The drum rotates at a speed of few hundred to several thousand rotations per minute to produce a relatively fast access time. To decrease access time, heads are sometimes located in sets around the periphery of the drum, so that a drum with 15 tracks may have 30 read/write heads divided in two sets of 15 heads, each set located 180° from the other. For very fast access time, there may be even more than two sets of heads.

The sizes and storage capacities of magnetic drums vary greatly. Small drums have capacity of less than 5,000 bytes. They generally have 20 to 30 tracks. Much larger drums having 500 to 1,000 tracks can store upto 100 megabytes.

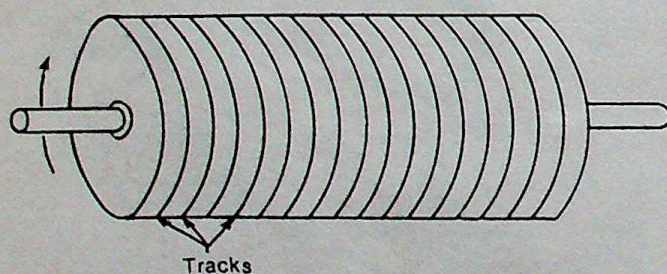


Figure 8.14. A magnetic drum storage

The larger drums normally rotate much slower than small drums and the access time obviously decreases as the drum speed increases. However, larger drums use several sets of read/write heads per track to maintain a relatively fast access time.

Magnetic drums have fast data transfer rates but are more limited in capacity than magnetic disks. They were an early means of primary storage. They were then used for online secondary storage when fast response was more

important than large capacity. For example, they were used to store mathematical tables, data, or program segments which were frequently needed during processing operations. Since drums are permanently mounted and cannot be replaced like disks, and since their storage capacity is relatively small, they are seldom used today.

## MASS STORAGE

Mass storage systems are storage systems that provide access to hundreds of billions of bytes of stored data. They combine the advantages of both tape and disk technologies. The storage medium is essentially a long, flexible plastic material upon which short strips of magnetic tape are mounted. These strips are then placed in cartridges and the cartridges are loaded into a storage device that is online to the CPU. The same read/write technique as with magnetic tape is used here to read and write data.

An example of such a device is the IBM 3850 Mass Storage System that can store 472 billion characters of data. The storage mechanism is shown in Figure 8.15. The Honeycomb storage compartments are used in this system to hold cartridges that contain data on tape.

The access times of mass storage systems are measured in seconds instead of milliseconds because the transport mechanism must move to retrieve the cartridge upon which the desired data is stored. It requires several seconds to locate the cartridge specified and then several more seconds are needed to transfer the data to a magnetic disk and then to the CPU. Thus an access time of several seconds is common for these storage systems. But a mass storage device has huge storage capacity and a very low cost per bit stored. Around 500,000 books of this size (the size of this book) can be stored in the IBM 3850.

Relatively slow access times limit the use of mass storage system in many applications. However, mass storage systems are a cost-effective alternative to other magnetic tape or disk storage in applications that require huge storage capacity and in which rapid access to data is not essential. When used for off-line storage, mass storage systems are often referred to as archival storage because they can store the very large volumes of historical or backup data that other systems can store.

## OPTICAL DISK

An optical-disk storage system consists of a rotating disk which is coated with a thin metal or other material that is highly reflective. Data recording is done by focusing a laser beam on the surface of the spinning disk. The laser beam is turned on and off at a varying rate because



tiny holes (or pits) are burnt into the metal surface of the disk along its tracks. In order to read the data, a less-powerful laser beam is focussed on the surface. This beam is strongly reflected by the coated surface and weakly reflected by the pits, producing patterns of on-off reflections that can be converted into electronic signals.

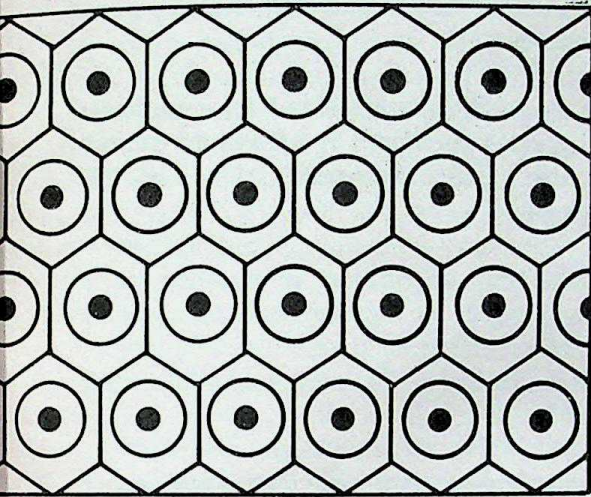


Figure 8.15. IBM 3850 Mass Storage System. The unit is a bee-hive of electronic activity. It can store upto 472 billion characters. This is accomplished by cartridges in each honeycomb cell that contain data on tape.

The storage density of optical disks is enormous, the cost is extremely low, and the access time is very fast. One small inexpensive disk will be able to store 25 to 30 reels of magnetic tape and any data on the disk can be accessed within a few milliseconds. Hence it is particularly suitable for the archival storage of vast amounts of data. Typical applications are image processing, geological survey data, medical publishing indexes, financial information files, and very large business data.

A serious shortcoming of currently available optical storage systems is that they are permanent storage devices. Once recorded, cannot be erased and hence the disk cannot be reused. Extensive research is being carried out to develop erasable optical disks and experts hope that it will be available within a decade.

## MAGNETIC BUBBLE MEMORY

The secondary storage devices that we have considered till now are electromechanical. Magnetic bubble memory (MBM), on the other hand, is an electronic secondary storage made with solid-state electronic chips

and have no moving parts. Magnetic bubble can be thought of as a tiny positively charged island in a sea of negatively charged film. They are formed by applying magnetic fields to thin sheets of certain magnetic materials, such as garnet crystal. The magnetic fields strengthen some regions in the material and weaken others. The strengthened regions break into isolated cylinders that resemble small positively charged islands surrounded by a sea of negative charges (Figure 8.16). Data is represented in bubble storage by the presence or absence of bubbles just as it is represented in punched paper tapes by the presence or absence of holes. The presence or absence of bubble corresponds to a 1 or 0 in the binary code. Bubble chips are non-volatile and retain their stored data even when the power is switched off.

Magnetic bubble memories are used as main memory in several microprocessor applications. They are also used as a low-cost alternative to magnetic disks. Portable terminals use bubble devices to store data until they can be transmitted to a larger system. Bubble chips are also used in telephone systems that redirect improperly dialed numbers, machine tools, robots, and military computers.

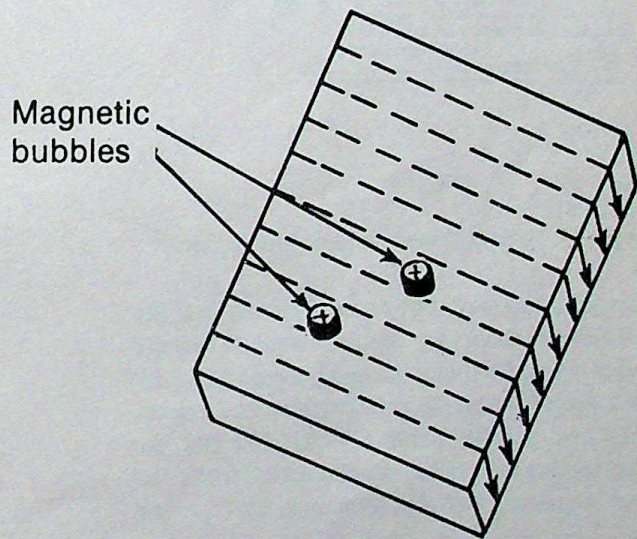


Figure 8.16. Formation of magnetic bubbles in a magnetic bubble memory.

## CHARGE-COUPLED DEVICE

Like magnetic bubble memory, a charge-coupled device (CCD) is also completely electronic, fabricated on semiconductor chips. It is a semiconductor that uses electrons within a metal-oxide semiconductor (MOS) crystal to store data. CCD is faster than MBM, is very compact, and may be inexpensive to produce in future. Unfortunately, however, CCD storage is volatile.



## A NOTE ON STORAGE HIERARCHY

As we have already learnt about the various types of storage devices used with computer systems, we will now briefly discuss the storage hierarchy that is applicable to most computer systems. A cost effective technique for the design of large computer systems is the use of a hierarchy of memory technologies. A typical storage hierarchy ladder is shown in Figure 8.17. It includes cache memory, main memory, secondary storage, and archival storage. As we move up the ladder, we encounter memory elements that have faster access time, higher cost per bit stored,

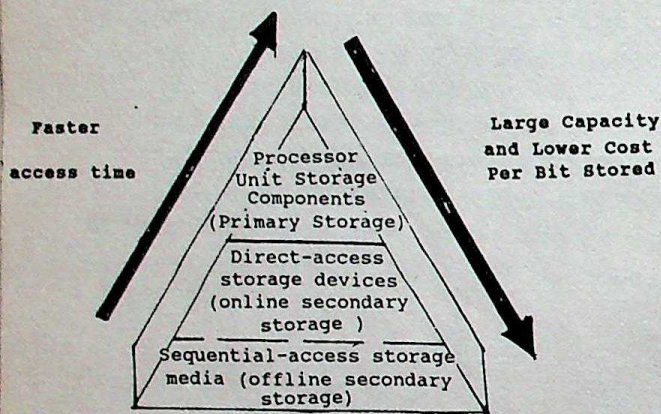


Figure 8.17. A typical storage hierarchy ladder.

and less capacity. A larger storage capacity, lower cost per bit stored, and slower access time are the results of moving down the ladder. Thus, CPU storage components generally have the fastest access times, the smallest storage capacity, and the highest cost per bit stored. On the other end, the archival storage elements have the slowest access times, the largest storage capacity, and the minimum cost per bit stored.

The cache memory which is plugged in between the CPU and the main memory is a very high speed semiconductor memory used to enhance the speed of main memory. The primary storage falls next in the memory hierarchy list. Secondary storage media such as magnetic disk memories make up the level of hierarchy just below main memory. Slower secondary storage devices, often referred to as archival storage, are at the bottom of the memory hierarchy. They are cost-effective for the storage of very large quantities of data when fast access time is not necessary.

## QUESTIONS

1. What is secondary storage ? How does it differ from a primary storage ?
2. Distinguish between a sequential access, a direct access, and a random access device. Give one example of each.
3. "The storage approach selected for a particular application is determined by the way the data are organized and processed." Discuss this statement.
4. Explain how information is recorded on a magnetic tape.
5. What is a parity bit ? How is it used for detecting errors ?
6. What is the data transfer rate for a magnetic tape system for which the tape density is 800 BPI and the tape speed is 200 inches per second ?
7. Discuss the relative advantages and disadvantages of magnetic tape and magnetic disk storage.
8. Discuss the advantages and disadvantages of tape cassettes and tape cartridges compared to conventional tape systems.
9. What is a disk pack ? Explain how information is stored in a disk pack.
10. What factors determine the storage capacity of disks ?
11. A disk pack consists of 6 disk plates. Each plate has 400 tracks and there are 50 sectors per track. If 256 bytes can be stored per sector then calculate the total number of bytes that can be stored in this pack.
12. List out the relative advantages and disadvantages of movable-head and fixed-head disk systems.
13. Explain seek time and latency for disk storage. What is the access time for a magnetic disk unit that has seek time of 20 milliseconds and a latency of 7 milliseconds ?



14. Fixed-head disk systems reduce total access time by avoiding either seek time or latency. Which is avoided and why?
15. What is "flying-head"? How does it help in increasing the life of a disk storage?
16. A magnetic drum has circumference of 50 inches and a packing density of 1000 bits/inch. If the drum has 40 tracks, how many bits can be stored on the surface of the drum?
17. What are the characteristics of a mass storage system? Why are they referred to as archival storage?
18. How are data recorded on and read from an optical disk? What is the major drawback of an optical disk storage?
19. What is a magnetic bubble? How are magnetic bubbles formed? Name some of the applications of magnetic bubble memories.
20. What is a memory hierarchy? Name the general classes of storage media that might make up a memory hierarchy.
21. For what type of application would you select magnetic tape? Magnetic disk? Archival storage?
22. Tell which secondary storage device would you choose for the following kinds of computer and why:
  - (a) Microcomputers
  - (b) Minicomputers
  - (c) Large computers.
23. Write short notes on the following :
  - (a) Punched paper tape
  - (b) Tape Cassettes and cartridges
  - (c) Floppy disk
  - (d) Winchester disk
24. Give the full form of the following abbreviations :
  - (a) CPI (d) CCD
  - (b) BPI (e) MOS
  - (c) MBM







## CHAPTER 9

# 9. INPUT - OUTPUT DEVICES

We have already seen in Chapter 2 that data and instructions must enter the computer system before any computation can be performed and the results of computation must be supplied to the outside world. Thus, a computer system can be proved to be useful only when it is able to communicate with its external environment. The input-output (abbreviated I/O) devices provide the means of communication between the computer and the outer world. They are also known as *peripheral devices* because they surround the CPU. Input devices are used to enter data into primary storage and output devices accept result from the primary storage to supply them to the users or to store them on a secondary storage device for future processing. There are some devices which are used for both the input and the output functions. The goal of this chapter is to familiarize the readers with the various types of I/O devices available for computer systems.

A wide variety of I/O devices are now available. For a particular application, one type of device may be more desirable than another. However, regardless of the nature of the I/O devices, special processors called *I/O interfaces* are required to convert the input data to the internal codes used by the computer and to convert internal codes to human

readable form while supplying the output. These I/O interfaces are also called channels or *I/O processors* (IOP). It is important to note that even the fastest of the I/O devices is very slow when compared to the speed of primary storage and CPU. The main reason for this is that the speed of I/O devices in most cases depends upon mechanical movement and the potential for improvement of such devices is limited. It has been difficult to produce I/O devices that can match the processor and storage speeds, and there is a constant demand for faster and faster I/O devices. We will now discuss at length the various types of I/O devices that are used for different types of applications.

### PUNCHED HOLE DEVICES

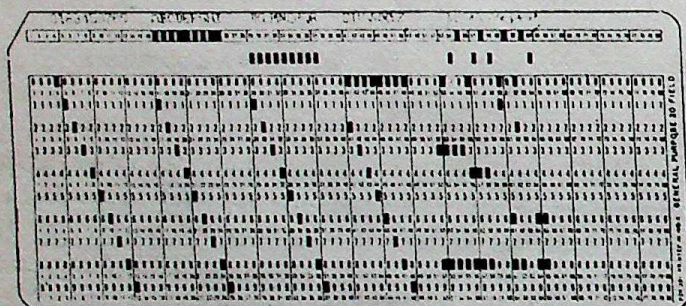
Punched hole devices are one of the oldest and cheapest I/O devices. Cards and paper tapes are the two most familiar punched hole devices. Data is recorded on these devices in the form of punched holes using some standard code. With the advent of better and faster I/O devices, the use of punched hole devices is diminishing day by day. However, many data processing centres still use them as a major input device.



## PUNCHED CARDS

Few years back, punched cards were the most widely used input medium for most computer systems. Even today they are extensively used in several computer centres. There are two types of punched cards. One has 80 columns and the other has 96 columns.

**The 80-Column Card.** It is 19.3 cms in length, 9.5 cms in width, and 0.018 cm in thickness. As shown in Figure 9.1, the card is divided from left to right into 80 vertical columns numbered 1 to 80. It is again divided into 12 rows numbered 12, 11, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9 from top to bottom. Each column of this card can represent one character. So a maximum of 80 characters can be represented by a single card. The digits 0 to 9 are represented by punching just one hole in the



| If only a numeric punch is in any Column, it represents whatever number is punched out | 12 punch and                                                                                    | 11 punch and                                                                                    | 0 punch and                                                                          |
|----------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------|
|                                                                                        | 1 --- A<br>2 --- B<br>3 --- C<br>4 --- D<br>5 --- E<br>6 --- F<br>7 --- G<br>8 --- H<br>9 --- I | 1 --- J<br>2 --- K<br>3 --- L<br>4 --- M<br>5 --- N<br>6 --- O<br>7 --- P<br>8 --- Q<br>9 --- R | 2 --- S<br>3 --- T<br>4 --- U<br>5 --- V<br>6 --- W<br>7 --- X<br>8 --- Y<br>9 --- Z |

Figure 9.1. An 80-column card.

corresponding row position. The alphabets A to Z are represented by a combination of two holes in two of the row positions. The top three rows 12, 11 and 0 are zone

punching positions and the rows 0 to 9 are numeric punching positions. A logical combination of zone and numeric punches is required to represent alphabets. For example, letter A through I are coded by using a 12-zone punch and numeric punches 1 through 9; letters J through R are coded by using a 11-zone punch and numeric punches 1 through 9 and letters S to Z are coded by using a 0-zone punch and numeric punches 2 through 9 respectively. Special characters are coded by punching one, two, or three holes. The coding system used to represent data in column cards is known as *Hollerith Code* after the name of Dr. Herman Hollerith who first used punched cards to handle the U.S. census data in 1889.

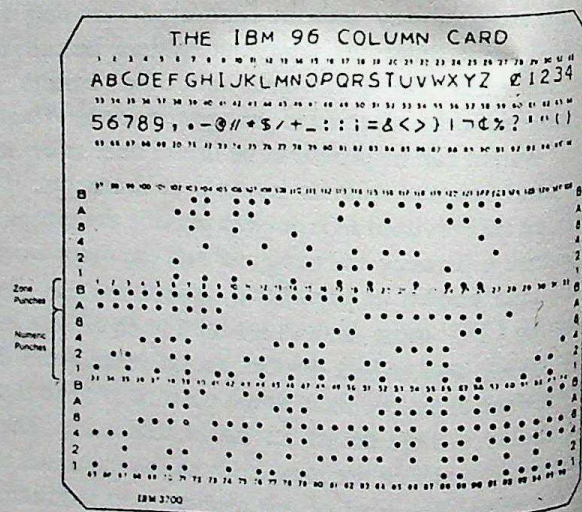


Figure 9.2. A 96-column card.

**The 96-Column Card.** As shown in Figure 9.2, it is only one-third the size of an 80-column card. The 96 columns are separated into three 32-column sections. The top three rows of the card which is not used for punching holes is used as a print area. These cards



round holes instead of rectangular holes of 80-column cards. Moreover, the standard 6-bit BCD code is used instead of Hollerith code for recording data on a 96-column card. Each of the 96 columns has six punch positions. The upper two are zone positions and the remaining four are numeric positions. The presence of a hole in a punch position indicates a 1 bit. It can be verified from the figure that the letter A is represented by the BCD code 110001.

**Card Punch machine.** It is used to transcribe data into cards. It resembles an elaborate typewriter which punches a set of holes in a card instead of printing characters on a paper. When the operator depresses a key, the correct combination of holes is punched in the column that is under the punch position at that particular moment. The machine contains a hopper in which blank cards are placed and a stacker in which the punched cards are stacked. The blank cards are automatically brought to the punch position from the hopper by pressing a special key meant for this purpose.

After punching, the cards are normally verified to ensure the correctness of data entered into the cards. A machine called a *verifier* is used for this purpose. The deck (group) of cards punched by an operator is placed in the hopper of the verifier and a second operator keys the same data. The verifier does not punch new cards, but compares what the second operator punches with the data already punched by the first operator. If a mismatch occurs in any card, the operator is notified to take corrective action. There are some machines which perform both the punching and the verifying tasks. In these machines, the first keying enters the data into an inbuilt storage section. A rekeying of the same data is then compared with the stored data. If the data match, a card is punched. In case of a mismatch, the operator makes necessary corrections and then the card is punched.

**Card Reader.** A card reader reads the information punched into a card, converting the presence or absence of a hole into an electrical signal representing a binary 0 or 1. Thus the holes in a card are converted into coded electrical pulses that the CPU can accept. The mechanism of a card reader is illustrated in Figure 9.3. As the card moves from the hopper to the stacker, it passes through a sensing station. At this sensing station, either wire brushes or photoelectric cells detect the presence or absence of holes. After the data is read at the first sensing station, the card passes through a second sensing station where it is read again. The data read at the two stations are then compared to verify the accuracy of the input operation. If the two reads match, the data is accepted by the card reader. If there is a mismatch, the card reader stops and an error message is given to the operator.

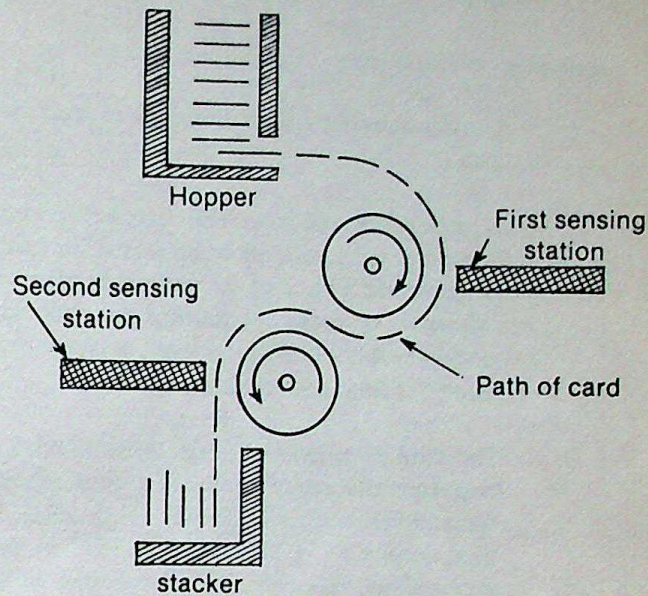


Figure 9.3. Mechanism of a card reader.

Card readers speeds vary from 300 to 2000 cards per minute. Thus for an 80-column card reader which operates at a speed of 2000 cards per minute, the data transfer rate will be  $80 \times 2000$  or 160,000 characters per minute. This data transfer rate is not even 1 percent of the data transfer rates of the magnetic devices such as magnetic disk and tape.

#### Advantages of Card Input

1. An erroneous instruction or data in a card-deck can be changed by simply throwing the incorrect card and replacing it with a correct card. This is not so easy in case of a magnetic tape, magnetic disk, or a punched-paper tape.
2. Unlike magnetic tape and magnetic disk, cards are humanly readable. Hence machine interpretation of data is not always necessary in case of cards.
3. In case of a card deck, insertion or removal of instructions or data is performed simply by inserting or removing individual cards. This is not so easy in case of a magnetic tape, magnetic disk, or a punched paper tape.

4. Cards are an old and reliable medium. They



remain a viable means of entering moderate amount of data into computer systems.

### Limitations of Card Input

1. Cards cannot be erased and used to enter new data.
2. A card deck needs careful handling because cards can be misplaced or separated from their proper deck. If a deck of cards falls down or somehow the cards get shuffled then the cards have to be laboriously rearranged in proper order before feeding the deck to the card reader.
3. The term *data density* refers to the number of characters that can be stored in a given physical space. Even when all columns are punched, the data density of a card is quite low. In most applications, however, not all columns will be punched and data density is thus further reduced.
4. Because of low data density, card files are often very bulky. Hence, cards are not suitable for the storage of voluminous data.
5. Folded, stapled, or mutilated cards are not acceptable by a card reader. So cards must be stored properly.

### PUNCHED PAPER TAPE

Perforated paper tape is also a very old media used as an I/O device for computers. Punched paper tape was already discussed in detail in Chapter 8 and hence its details will not be repeated here.

**Tape Punch Machine.** It resembles a typewriter. The keyboards of many tape punch machines are identical with the keyboards of conventional typewriters. When a key on the tape-punch keyboard is depressed, the binary-coded symbol for the character selected is punched into the tape, and the tape then advances to the next line. In addition to punching holes for a character on the tape, most tape punch machines also print the punched character on a separate piece of paper just like a typewriter. This printed copy of the data punched on a tape is called the *hard copy* which is used for manual detection of errors, if any, that might have occurred during punching.

**Paper Tape Reader.** It reads the information punched into a paper tape and converts the coded information into electrical signals that the CPU can accept. It operates in a similar manner to the card reader. As the

tape passes from the sensing station, the holes are sensed either by sensing pins or photoelectric cells. Readers using photoelectric cells are much faster as compared to those using sensing pins. Typical speeds of paper tape readers are in the range of 250 characters per second to 1000 characters per second.

### Advantages and Limitations of Paper Tape

1. Paper tape has greater data density than punched cards.
2. Unlike punched cards, there is no wastage of columns with punched tapes.
3. Paper tape is cheaper than cards, and paper tape equipments are less expensive than card machines.
4. Tape accuracy is harder to verify than card accuracy, and errors that are detected are not so easy to correct.
5. It is also difficult to delete or add information in the middle. Splicing is often required.
6. Like cards, paper tapes are easily torn and mutilated.
7. Like cards, it is not possible to erase data from a paper tape and to enter new data on the same tape.

## MAGNETIC MEDIA DEVICES

Magnetic disks, diskettes, and tapes can record data as output from primary storage and can also serve as input devices returning the data to primary storage. In fact, all the secondary storage devices discussed in Chapter 8 at magnetic tape, tape cassettes and cartridges, magnetic disk floppy disk, winchester disk, magnetic drum, magnetic bubble, optical disk, etc. are both input and output devices.

Because the magnetic media devices have already been described in detail in Chapter 8, we will not discuss them here.

## PRINTERS

Printers are the most commonly used output devices that can be found in almost all computer centers. They are the primary output devices used to prepare permanent documents in human-readable form (hard copy). There are several types of printers that are designed for different



types of applications. Depending on their speed and approach of printing, printers are classified as *character printers*, *line printers*, and *page printers*. In addition, there is another classification according to which printers are of two types - impact and nonimpact. *Impact printers* use the familiar typewriter approach of hammering a typeface against paper and inked ribbon. *Nonimpact printers* do not hit or impact a ribbon to print. They use thermal, electrostatic, chemical, and inkjet technologies.

## CHARACTER PRINTERS

Character printers print only one character at a time. A typewriter is an example of a character printer. Three of the most commonly used character printers are described below.

**Letter-Quality Printers.** These printers use a printwheel font known as a daisy wheel (Figure 9.4). Each petal of the daisy wheel has a character embossed on it. A motor spins the wheel at a rapid rate. When the desired character spins to the correct position, a print hammer strikes it to produce the output. Thus, daisy wheel printers are impact printers. Their output resembles a typed output and they are noted for their print quality. The speed of these letter-quality printers normally ranges from 10 to 50 characters per second.

**Dot-Matrix Printers.** These printers print each character as a pattern of dots. The print head comprises a matrix of tiny needles, typically seven rows with nine needles in each (9 x 7 matrix), which hammers out characters in the form of patterns of tiny dots. The shape of each character, i.e. the dot pattern, is obtained from information held electronically in the printer. Figure 9.5 shows a dot-matrix printer and Figure 9.6 shows the dot pattern used to produce the characters.

The print quality of a dot-matrix printer is inferior to that of a daisywheel printer. But dot-matrix printers are generally faster than letter-quality printers - in the range of 10 to 250 characters per second. They are also less expensive than a daisywheel printer and hence if the quality of output is not an important factor then the dot-matrix printer is used with most microcomputer systems. There is an additional advantage of dot-matrix printers over letter-quality printers in the sense that dot-matrix printers do not have a fixed character font. So they can print any shape of character that a programmer can describe. This allows for many special characters, different sizes of print, and the ability to print graphics such as charts and graphs.

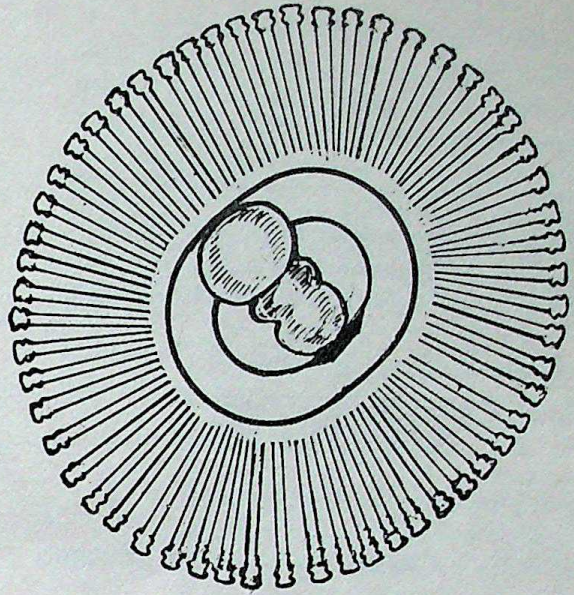


Figure 9.4. A daisy wheel.

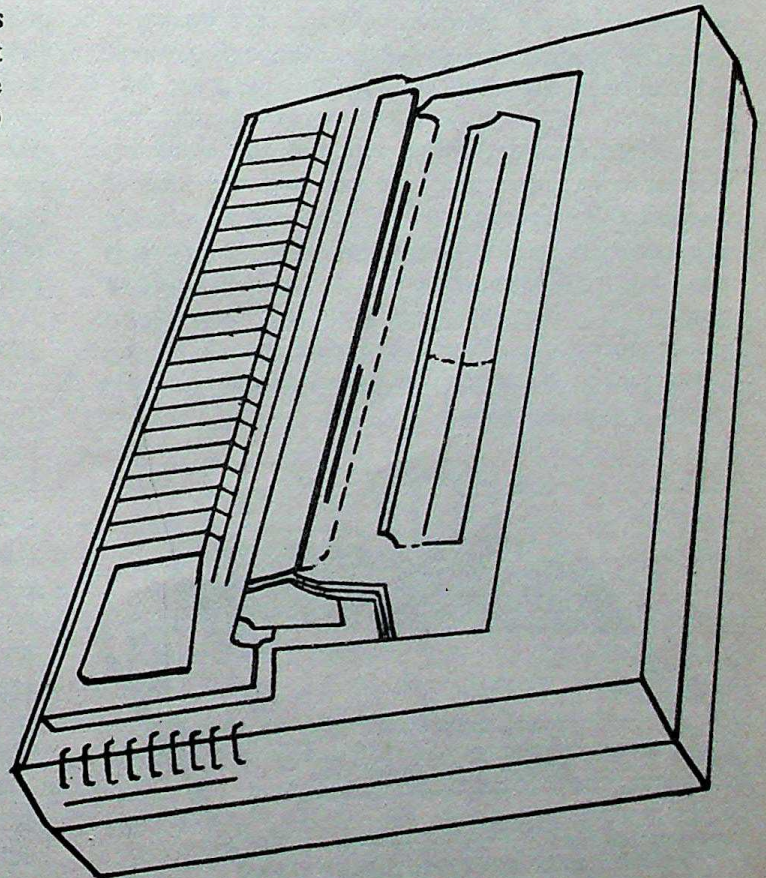


Figure 9.5. A dot-matrix printer.



## LINE PRINTERS

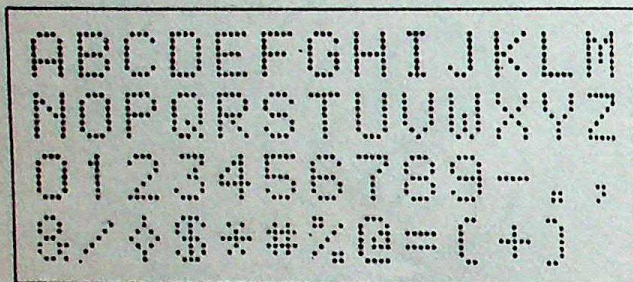


Figure 9.6. Character pattern for a dot-matrix printer.

**Ink-Jet Printers.** These are nonimpact character printers based on a relatively new technology. They print characters by spraying small drops of ink onto paper. Special type of ink having a high iron content is used. Droplets of ink are electrically charged after leaving a nozzle. The droplets are then guided to the proper position on the paper by electrically charged deflection plates. Ink-jet printers produce high quality output because the characters are formed by dozens of tiny ink dots. Moreover, they are quiet (nonimpact) and can form any kind of character. The document printed may contain multiple character styles and a variety of type sizes. Some models also allow for different colours of ink for multiple colour printing. Of course, inkjet and other nonimpact printers cannot produce multiple copies of a document in a single printing which is possible with impact printers. An inkjet printer is shown in Figure 9.7.

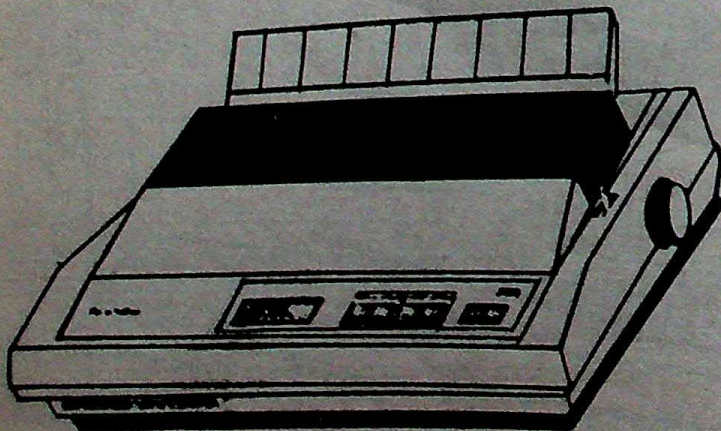


Figure 9.7. An inkjet printer.

Line printers are impact printers used with medium and large computer systems for producing high volume paper output. They are fast printers having speeds in the range of 300 to 2500 lines per minute. Their printing speeds are such that to an observer they appear to be printing a line at a time and hence the name 'line printer'. There are normally 132 or 136 print positions per line. Some line printers even have a line length of 120 characters and a few have a line length of 144 characters. Character sets vary in content and size for different line printers. The 64-set has more special characters than the 48-set and the 96-set prints in lower-case (small) letters as well as upper-case (capitals). The output efficiency of a line printer can be increased by using multiple-part papers (pages with carbon paper inserts) when more than one copy of the output is required. Drum printer and chain printer are the two most commonly used line printers.

**Drum Printers.** As shown in Figure 9.8, a drum printer consists of a solid, cylindrical drum that has raised characters in bands on its surface. There are as many bands as there are printing positions. Each band contains all the possible characters. The drum rotates at a rapid speed. For each possible print position, that is, opposite to each band of the drum, there is a print hammer located behind the paper. These hammers strike the paper, along with the inked ribbon, against the proper character on the drum as it passes. One revolution of the drum is required to print each line. This means that all characters on the line are printed at exactly the same time, but the time required to print the entire line is fast enough to call them line printers. Typical speeds of drum printers are in the range of 300 to 2000 lines per minute.

**Chain printers.** They use a rapidly moving chain called a *print chain*. Each link of the chain is a character from a font. A typical print chain is shown in Figure 9.9. Instead of a chain, some models of printers use a metal band having raised print characters on it. For each possible print position, there is a print hammer located behind the paper. As the print chain or the band rotates, the properly timed print hammers strike the paper, along with the inked ribbon, against the proper character on the chain/band as it passes. In order to enhance the speed of the chain printers, the character set is repeated several times on the chain. Hence it is not necessary to wait for the chain to make a complete revolution to position the desired character in the correct print position. A standard character set is of 48 characters. So chain printers that use the 64-set or the 96-set repeat the characters fewer times than the standard set. This increases the time to position the desired character and therefore the printing speed.



reduces the effective print speed of the printer. Speeds of chain printers range from 400 to 2500 characters per minute.

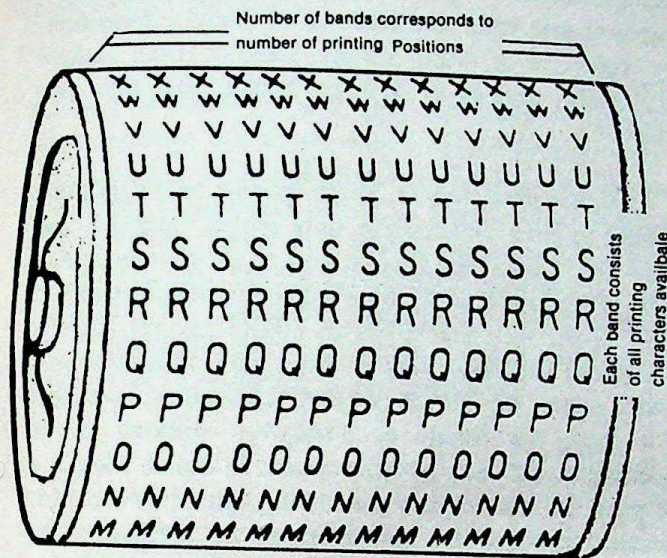


Figure 9.8. The solid cylindrical drum of a drum printer.

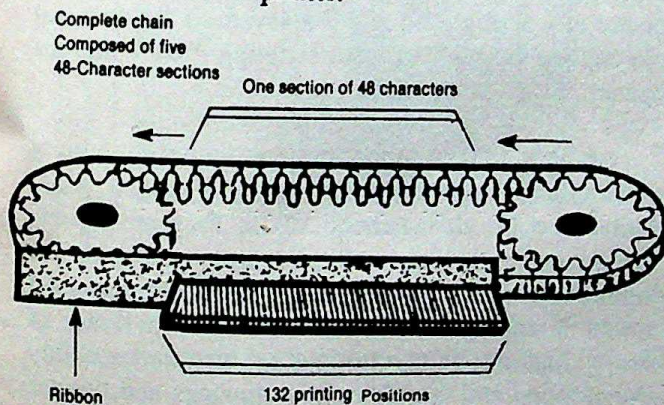


Figure 9.9. The print chain of a chain printer.

## PAGE PRINTERS

These are very high speed nonimpact printers which can produce documents at speeds of over 20,000 lines per minute. (That is fast enough to print this entire book within five minutes !). Electronics, xerography, lasers, and other technologies have made these high-volume systems possible. These techniques, called *electrophotographic techniques*, have developed from the paper copier technology. Page printers can produce pages of output at a rate equal to a paper copier. Each page produced on these printers is an original one since there are no carbon copies. Because of their extremely high cost, these printers are economical only when hundreds of thousands of pages are to be printed each month.

## KEYBOARD DEVICES

Keyboard devices are online devices that are used for entering data directly into a computer. Data is entered into the computer by pressing a set of keys available with these devices. Keyboard devices are becoming more and more popular day-by-day. They are extensively being used today both as input and output devices. They are available in a wide variety. Few of the most commonly used ones have been described below. These and other online processing devices have the following characteristics :

1. No intermediate data recording media is required with these devices. They are able to enter data directly into the computer.
2. They can be situated far away from the main computer room. This allows these devices to be located at or near the data source.
3. These devices allow the computer to be used in interactive mode. That is, they allow the user to enter data through the keyboard while the program is running and to get immediate answers from the computer.
4. They are economical for handling a lower and/or more irregular volume of input data.

## VIDEO DISPLAY TERMINALS

Video display terminals (VDTs) are the most popular I/O devices used today in direct-access processing applications. A typewriter like keyboard, as shown in Figure 9.10, is used to enter data into the computer, and a cathode ray tube (CRT) that looks like a television screen is used to display the input data as well as the messages and processed output from the computer. The terminal consists of a small memory known as a buffer. The size of this buffer is normally equal to the total number of characters (usually 1920 - 24 rows x 80 columns) that can be displayed at a time on the terminal screen. Each character entered through the keyboard is stored in the buffer and is also displayed simultaneously on the CRT. The data is not sent on to the computer until the operator presses an enter/transmit key on the keyboard. This gives an opportunity to the operator to proofread the data being entered by reading the data displayed on the screen. A small square or underscore character, called a *cursor*, indicates the operator where the next character to be keyed will be displayed on the screen. Any keystroke error can be easily corrected by moving the cursor to the erroneous character and rekeying the data. Thus, error correction process is very easy as long as the data has not been transmitted on to the



computer. A typical VDT is shown in Figure 9.11. VDTs facilitate instantaneous display of operation and they are totally silent in their operation. The primary disadvantage is that no "hard copy" or record of what is typed by the terminal operator or output displayed by the computer remains when the terminal is switched off. However, it is normally possible to add a printing device which can be switched on to provide a hard copy of the display when it is needed. Thus more expensive VDTs include cameras to photograph the scope face, small printer, and tape cassettes to record data.

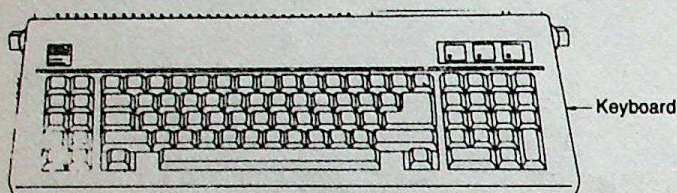


Figure 9.10. A keyboard of a video display terminal.

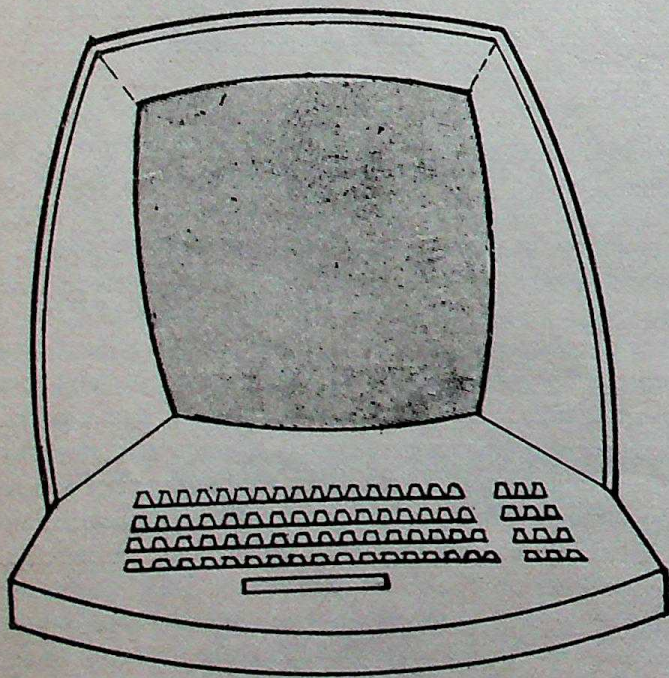


Figure 9.11. A typical video display terminal

Initially used in applications where information is required quickly, for example in airline reservation systems, bank counters, etc. where speed is the essence in handling customer enquiries, VDTs are now widely used for general data entry and retrieval of stored information. They come in a wide variety ranging from simple alphanumeric devices to complex graphic and light pen devices.

*Alphanumeric VDTs.* These are low cost VDTs that are used to enter and retrieve only letters, numbers, and special characters. They may be used by shop employees to update production records, by word processing operators to create letters, by bank tellers to update savings account records, etc. The availability of airline seats, railway berths, or hotel rooms, the current stock of various items available in a shop, the details of employees working in an organisation, etc., can be kept current in online files in a computer and then a query entered through the keyboard is instantly answered by the computer by displaying the information on the terminal screen. Alphanumeric display terminals are also used by programmers to directly prepare and/or maintain computer programs. New programs can be directly entered and old programs already stored inside the computer can be retrieved and displayed on the terminal screen and can be modified before storing it back into the computer.

*Graphic VDTs.* Video display terminals with a graphic display capability are normally much more expensive than the alphanumeric VDTs. They are capable of displaying graphics and diagrams as well as alphanumeric characters and are used particularly as an aid to design. Graphic devices provide not only a means of displaying high-resolution drawings but also the capability of manipulating and modifying the drawings and designs displayed on the terminal screen. Different views of a design such as a building can be displayed and it is also possible to alter the scale of presentation so that either the entire design or a small section of the design can be properly examined. Using coloured graphic devices, it is also possible to display different portions of the design in different colours and to highlight particular portions of the display.

Graphic devices are currently being used as an aid in the design of cars, ships, aircrafts, buildings, highways, electronic circuits, etc. They are also being used to present business, financial, and operating data in graphical form so that managers can spot trends and relationships and can then make faster and better decisions. Pages of computer printouts detailing the performance of product lines and sales divisions can be replaced by a few colourful charts, graphs, and maps which are very helpful for higher



management in making future policy decisions. A copying device can be used in conjunction with a graphic VDT to provide hard copy of any display. A typical graphic display is shown in Figure 9.12.

**Light Pen VDTs.** A light pen is an input device attached to some video display terminals. It is penlike photo-sensitive instrument which can sense a position on the CRT when the end of the pen is held against the screen. It consists of photocell placed in a small tube. When the user moves the pen over the screen, the pen is able to detect the light coming from a limited field of view. The light from the CRT causes the photocell to respond when the pen is pointed directly at a lighted area. These electrical responses are transmitted to the computer,

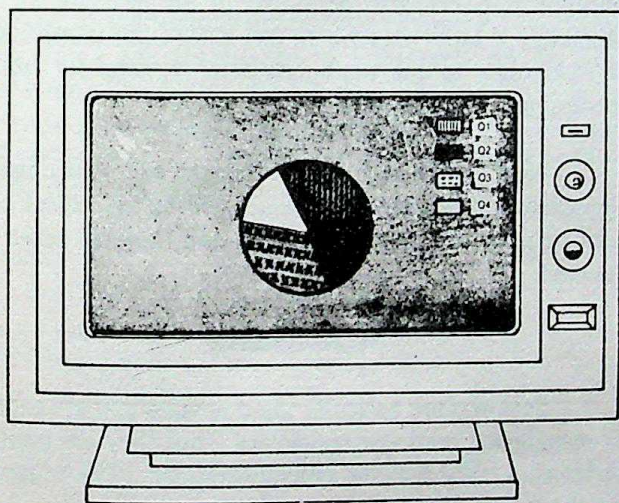


Figure 9.12. A typical graphic display.

which is able to determine that part of the displayed item which is triggering the photocell response.

By sensing the position on the screen being touched by the light pen, the user inputs data to the program that is currently active in the computer. For example, the screen display might have a set of command choices such as COPY, DELETE, EDIT, HELP, INSERT, PRINT, SORT. If the user touches by the light pen that portion of the terminal screen where PRINT is displayed, then he is inputting PRINT command as his choice to the computer.

Similarly, there are various other applications of light pen devices.

Light pens are also used in conjunction with graphic terminals to help in computer-aided design (CAD). The user "draws" directly on the screen with the light pen. By using the light pen and a keypad attached to the terminal, the user can select different colours and line thickness, can reduce or enlarge drawings, and can add or erase lines. This approach is commonly used by engineers to modify designs which are displayed on a CRT during the design and development process.

### TELEPRINTER TERMINALS

A teleprinter terminal is very similar to a video display terminal except that it uses a printer in conjunction with a keyboard instead of a CRT. Thus it provides 'hard copy' (printed record) of the input, system information and program results. However, these printers are character printers that print one character at a time and therefore teleprinter terminals are slower than video display terminals. A teleprinter terminal is shown in Figure 9.13.

Several small portable teleprinters are being used today by salespeople, managers, newspaper reporters, engineers, and others on the move. The portable teleprinter, in effect, lets these people take a computer with them wherever they go and enables the collection of data at the place where it is generated. This avoids the delay and cost of data preparation at the computer site. For example, a route salesman calling on retailers can carry a terminal through a customer's store and can directly key in the product number and quantity needed into the terminal when items are found to be short of supply. Similarly a manager can carry a terminal along with him on a business trip and keep up with office work whenever he finds some free time.

### POINT-OF-SALE TERMINALS

A point-of-sale (POS) terminal is basically an electronic equivalent of a cash register. It can perform all the functions of a cash register with an added advantage of capturing more data than a cash register. Some POS terminals are offline devices that record data on cassette tapes. These tapes are then collected and are carried to computer site where the actual processing is done. Thus, these POS devices do not avail the facility of a computer directly. However, most of today's POS terminals function online and are directly connected to a computer. These POS devices process transactions immediately while the customer is making purchase. For example, under computer control, the terminal may display the transaction data and then print on a receipt that shows the total



amount of the purchase including taxes. If a credit card is used to complete the transaction, the number on the credit card can be entered into the computer to update the customer's credit account. Transaction data can also be used to update inventory records and provide sales analysis information to managers. Till now POS devices are not in common use in India.

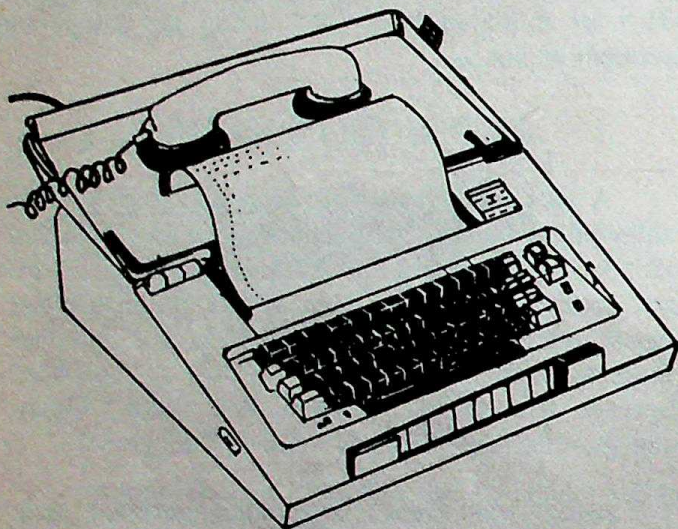


Figure 9.13. A teleprinter terminal.

### INTELLIGENT TERMINALS

Terminals are extensively being used today as online, remote data entry devices. Depending on their local processing capability, these terminals are classified as dumb, smart, and intelligent.

A simple, low-cost teleprinter or alphanumeric display device is generally classified as a *dumb terminal*. These units have a keyboard for input, a means of communicating with the CPU of the central-site computer, and a printer or screen to receive output. Dumb terminals need not appear dumb to the user. Because of the rapidity with which the central-site computer can communicate back to the data entry station, many applications are not affected if no actual processing is done locally. However, dumb terminals transmit all data directly to the central-site computer. This type of data is called "dirty" data because errors are transmitted along with good data. The errors are then identified by the CPU of the central-site computer and processing could be delayed until "clean" data are obtained from remote users. This type of delay, accompanied by the need to process the data over again, can be expensive.

Smart terminals have additional features. They usually have a microprocessor (a CPU) and some internal storage in addition to I/O capabilities. They have local data editing capability and the ability to consolidate input data prior to sending them to the CPU of the main computer-site. This means that preliminary processing of input data can take place locally and that the data can be cleaned up before it is transmitted to the central-site computer. However, smart terminals cannot be programmed by users.

An intelligent terminal is a small computer by itself. It consists of a built-in microcomputer that can be programmed by users. These programmable terminals have extensive local data processing power. Large working files can be maintained locally and small data processing jobs can be handled by the terminal itself without the need to interact with the CPU of the main computer system. Online secondary floppy disk storage devices are often used during the processing of these small jobs. Programmed error-detection tests can also be stored in the terminal for the validation of input data which can ensure the transmission of clean data to the main computer system for further processing.

With the fast reduction in the cost of microcomputer components, most future terminals will be intelligent or at least smart to help make data entry and retrieval easier for the users.

### SCANNERS

Scanners are basically input devices that are capable of recognising marks or characters. Thus they are used for direct data entry into the computer system. The following are the characteristics of scanners:

1. They eliminate some of the duplication of human effort required to get data into the computer. Human beings do not have to key the data.
2. The reduction in human intervention improves data accuracy and can increase the timeliness of the information processed.
3. Since scanners are direct data entry devices, they demand high quality of input documents. Documents that are poorly typed or have strikeovers or erasures are normally rejected.
4. With these devices, form design and its specification may become more critical than in the case when people key the data from the forms.
5. Most of these devices are not economically feasible unless the daily volume of transactions is relatively high.

Though not very popular in India, scanners are



extensively used in advanced countries like U.S.A. The two major type of scanners are optical scanners and magnetic-ink character recognition devices.

### OPTICAL SCANNERS

As the name implies, the technique used in these devices for the recognition of marks/characters involves a light source and light sensors. These devices are capable of interpreting handmade marks and characters, machine-printed characters, and special bar code. Thus the common optical scanner devices are optional mark readers (OMR), optical character readers (OCR) and bar-code readers.

**Optical Mark Readers.** These scanners are capable of recognising a prespecified type of mark made by pencil or pen. For example, many students might have appeared in objective type tests where they had to mark their answers to questions on a special test-scoring sheet by darkening a square or circular space by a pencil to indicate their correct choice out of the various alternatives (Figure 9.14). These answer sheets are directly fed to a computer for grading with the use of an optical mark reader. The use of OMR is not limited to the grading of objective type tests. In fact, any input data that is of a choice or selection nature can be recorded for OMR input.

#### Questions:

1. The capital of India is:
  - (a) Bombay
  - (b) Delhi
  - (c) Calcutta
  - (d) Madras
2. The binary equivalent of decimal 4 is :
  - (a) 101
  - (b) 111
  - (c) 001
  - (d) 100

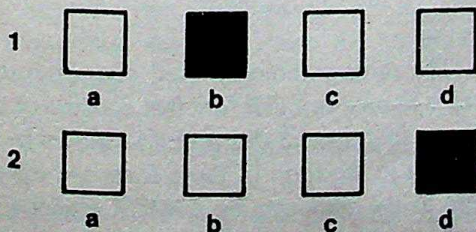


Figure 9.14. Sample questions of an objective test and its answer sheet that can be read by an optical mark reader to grade the test.

The actual technique used by an OMR device for recognition of marks involves focussing a light on the page being scanned and detecting the reflected light pattern from the marks. Pencil marks made with a soft lead pencil (high

graphite content) will reflect the light.

**Optical Character Readers.** These scanner devices are capable of detecting alphabetic and numeric characters printed on paper. These characters may be either typewritten or handwritten. In case of handwritten characters, special care has to be taken to ensure that the characters are of standard size, lines making up the characters are connected, and no stylish loops etc. are used. On the other hand, if the characters are typewritten, they must be typed using a special type font called an OCR font. A wide range of fonts, using ordinary inks, can now be accepted by OCR devices. The standard fonts used are OCR-A (American standard) and OCR-B (European standard). OCR-A fonts are shown in Figure 9.15. Most brands of typewriters are available with an OCR character set. Some OCR devices can also accept computer print-out and complete pages of typed text.

OCR devices examine each character as if it were made up of a collection of minute spots (pixels). Once the whole character has been scanned, it is compared with the characters (standard fonts) the machine has been programmed to recognize. Whichever pattern it matches, or nearly matches, is considered to be the character read. If the scanned character does not match satisfactorily with any of the fonts, it is rejected.

OCR devices are expensive and are used only for large-volume processing applications. For example, the computer-printed bills sent to customers by many public utilities, credit card companies, and other businesses in U.S.A. are prepared with characters that can be read by OCR devices. When customers make their monthly payments, they are instructed to return the bill or a remittance stub with their checks. These documents are then entered directly into optical readers to update accounts-receivable records. Little or no human keying is needed. Other large-volume applications of OCR devices include the reading of zip codes by the U.S. postal service, the reading of passenger tickets and freight bills by airlines, and the processing of social security forms and motor vehicle registrations by governments.

**Bar-Code Readers.** Data coded in the form of light and dark lines or bars are known as bar codes. Bar-Codes are used particularly by the retail trade for labelling goods and by supermarkets for labelling shelves and in stock control. They are also used for numbering books in public libraries so that when a book is borrowed or returned, it can be recorded using a computer.

Bar-Code reader is a device used for reading (decoding) bar-coded data. Bar-code reading is performed





Figure 9.15. OCR-A fonts.

by a laser-beam scanner which is linked to a computer. The laser-beam is stroked across the pattern of bits that is recorded as the input data.

The most widely known bar code is the *Universal Product Code* (UPC) which now appears on almost all retail packages in U.S.A. These bars are decoded as 10 digits. The first five of these digits identify the manufacturer or supplier of the product and the second five digits identify a specific product of the manufacturer (Figure 9.16).

The UPC is designed to simplify customer checkout and retail store inventory management. For example, when bar-coded items are received at a merchant's automated checkout stand, they are decoded by a bar-code reader and the data is transmitted to a computer that looks up the price possibly updates inventory and sales records, and forwards price and description information back to the check stand for output on the sales receipt.

### MAGNETIC-INK CHARACTER RECOGNITION

Magnetic-ink character recognition (MICR) devices were developed to assist the banking industry in processing the tremendous volume of checks being written every day. A special type of cheque is used in banks that employ MICR devices. The bank's identification code and the customer's account number is pre-printed (precoded) on all these cheques with a special ink that contains magnetizable particles of iron oxide (Figure 9.17). When a filled-in cheque is presented at the bank, a bank employee encodes the amount in the lower right corner. This cheque is then processed using a MICR device.

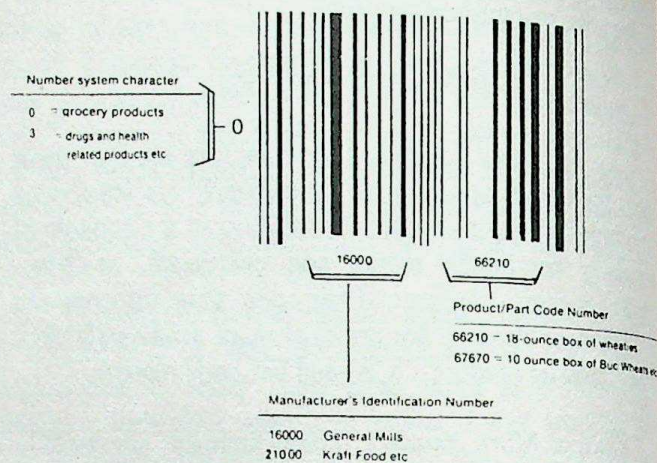


Figure 9.16. UPC bar code.

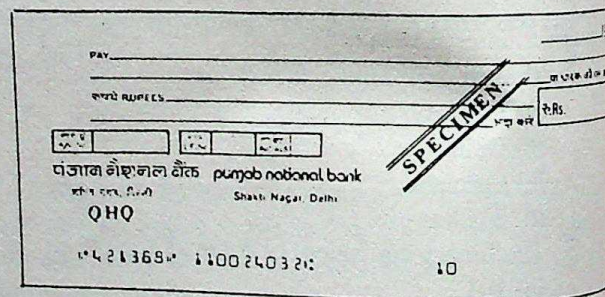


Figure 9.17. A bank cheque that employs MICR device.

The most commonly used character set by MICR devices is known as E13B font that consists of the numbers 0 to 9 and four special characters as shown in Figure 9.18. Coded data in the form of these fonts are transferred from cheques to the computer by a MICR reader-sorter. As cheques enter the reading unit, they pass through a magnetic field which causes the particles in the ink to become magnetized. Read heads then interpret the characters by examining their shapes using a 7 x 10 matrix and determines, from the response of the segments of the matrix to the magnetic ink, which of the characters has passed under the reader's head. The sorter is basically used to sort the cheques into different pockets according to their identification code numbers.



The following are some of the advantages and limitations of MICR system :

1. Even roughly handled, folded, smeared, and over stamped cheques can still be read with a high degree of accuracy.

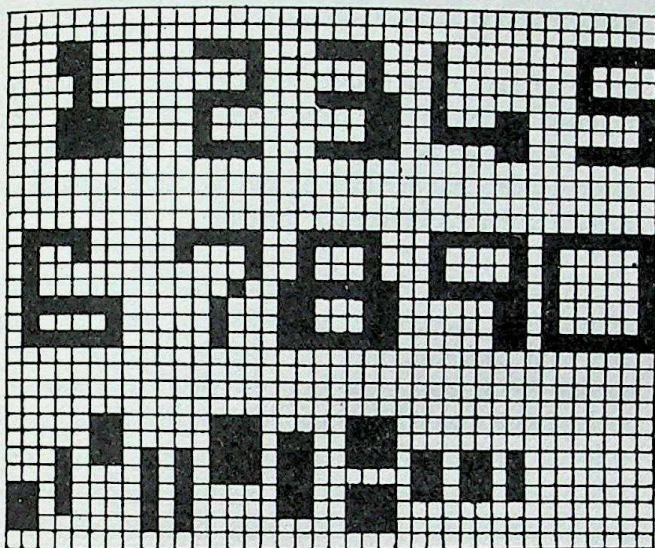


Figure 9.18. The E13B font (character set) used by MICR devices.

2. MICR devices speed up data input for the banking industry because cheques can be directly fed into the input device.
3. The characters coded on the cheques using magnetic ink can also be easily read by people.
4. The main limitation of MICR is that only the 10 digits and 4 special characters needed for bank processing are used. No alphabetic characters are available.
5. Special type of magnetic ink is required for encoding the characters to be read by the machine.

## OTHER DEVICES

In addition to the already discussed I/O devices, there are some I/O devices which are not so common in use. These are specialized devices used for some special type of application by few data processing centres. Such type of devices will be discussed in this section.

### COMPUTER OUTPUT MICROFILM

Computer output microfilm (COM) technology is used to record computer output information as microscopic

filmed images. Thus COM is basically an output device that records information on a sheet or roll of microfilm. A sheet of film measuring 4 x 6 inches is called a microfiche. (Fiche is a French word meaning card and is pronounced as "fish"). A typical microfiche is capable of holding 270 frames (pages of information) and some ultrafiche systems can even store 1,000 frames in the same space. Rolls of 16 or 35 millimeter film packaged in cartridges are also used for COM recording. COM recording process produces characters that are 48 or more times smaller than those produced by conventional printers. Thus a single microfiche can hold approximately 300 times as much information as a standard computer printed page.

The COM recording technology consists of a microfilm recorder that receives information that are normally stored on magnetic tapes. Recorders may also receive the information directly from the CPU. The recorder in turn projects the characters of output information on to a CRT screen. A high-speed camera, in-built into the system, now takes a picture of the displayed information. In some systems, the recorder processes the film from the camera and in other systems, a separate automated film developer is used. Film duplicators can make as many copies of the developed film as needed.

A special device known as *microfilm reader* (Figure 9.19) is used to view the information recorded on the sheets or rolls of film. In some COM systems, users must locate and then manually search through the film cartridge or card to find the desired information. However, in computer assisted retrieval systems, the correct document from the thousands of pages that may be recorded on the film can be automatically picked up in a matter of seconds simply by specifying the preassigned index that corresponds to that particular document. The microfilm reader operates on a 'back projection' principle displaying a frame at a time on a translucent screen, typically about A4 size. Some COM systems also use a reader-printer combination to produce a hard copy of what is presented on the screen. A COM system is ideal for use in applications where there is a large amount of information to be retained which is required only for manuals, industrial catalogues and archives. Companies often need to retain records of such things as bills and invoices for a number of years before destroying them. COM provides an easy and compact way of retaining the information, of retrieving it in a matter of seconds using a compact desk top reader, and is ideal when multiple copies of reports or information are required. Banking and insurance companies, government agencies, public utilities, and many other types of organizations are regular users of



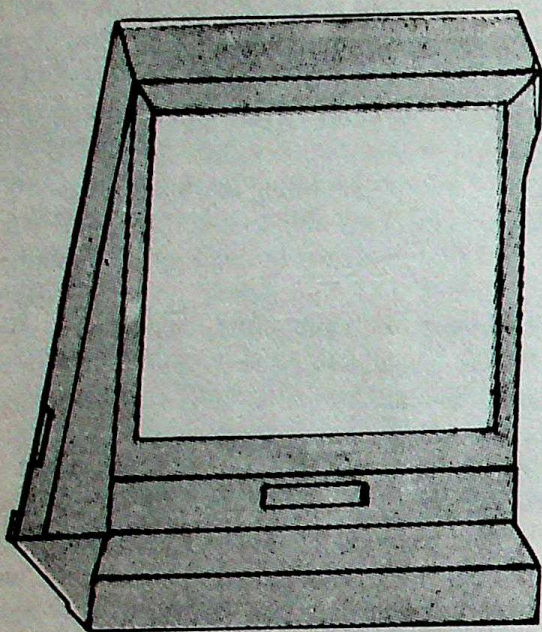


Figure 9.19. Microfiche reader.

The advantages and limitations of COM are as follows:

1. The speed of recording of a COM system is 20 to 25 times faster than the average line printer. Thus, a single COM recorder can do the work of a dozen line printers.
2. The microfilm equivalent of thousands of pages of computer output can be stored in a small drawer. Thus the mailing and storage costs of filmed documents are much less than paper documents.
3. Microfilm output is even less costly as compared to printed output. The cost of the paper needed to print a 1000 page, three-part report is about 30 times greater than the cost of the film needed to do the same job.
4. In case of filmed output, it is possible to record drawings or pictures as well as alphanumeric character texts.
5. COM systems are relatively expensive to install and their high initial investment is justified only in case of high volume workload.
6. Because of retrieval delays and the lack of facility to write notes on the margins of the reports, many people feel uncomfortable when using a COM system.

## DIGITIZERS

A digitizer is an input device that converts graphic and pictorial data to digital (binary) form which can be directly fed and stored inside a computer. There are two types of digitizers: rectangular-coordinate or flatbed digitizer and image-scan digitizer.

In case of a flatbed digitizer, the drawing to be digitized is spread and fixed over a rectangular flatbed table. A mechanism is now moved over the surface of the drawing that scans the drawing and produces signals related to the X and Y coordinates of the table.

Image-scan digitizers scan and reproduce entire drawing and photographs automatically. They are costlier and more powerful than the flatbed digitizers and are capable of digitizing not only the shape and size of the drawings but also the varying intensities on a gray-to-black scale at different points of the drawings. Thus, flatbed digitizers are mainly used to digitize simple drawings, graphs, charts, etc. and image-scan digitizers are used to digitize more complex pictures and photographs.

## PLOTTERS

A plotter is an output device used to produce hard copies of graphs and designs. Plotters are basically of two types - drum and flatbed. In case of a drum plotter, the paper on which the design has to be made is placed over a drum that rotates back and forth to produce vertical motion. The mechanism also consists of one or more pen holders mounted horizontally across the drum. The pen(s) clamped in the holder(s) can move to produce horizontal motion. Under the control of the computer, the drum and the pen(s) move simultaneously to produce the designs and graphs. Since each pen is program selectable, pens having ink of different colours can be mounted in different holders to produce multi-coloured designs.

As the name implies, a flatbed plotter plots on papers that are spread and fixed over a rectangular flatbed table. In this type, normally the paper does not move and the pen-holding mechanism is designed to provide all the motion. Here also provision is there to mount more than one pen in the pen(s) holding mechanism. Normally pens with ink of different colours are used for multi-coloured plotting. The plot size is restricted by the area of the bed. Some may be as small as A4 size whilst some very large beds used in the design of cars, ships, aircrafts, buildings, highways, etc. can be up to 20 ft. by 50 ft. Some plotters are also designed to etch plastic or metal plates.

Plotters are normally very slow in motion because of



excessive mechanical movement required during plotting. Hence there is a great mismatch between the speed of the CPU and the speed of a plotter. Because of this reason, in most cases, output is first transferred by the CPU on to a tape and then the plotter is activated to plot the design from the information on the tape. However, in case of a computer system dedicated to design work, the CPU may send output directly to a plotter.

## VOICE RECOGNITION AND RESPONSE DEVICES

These I/O devices have made it possible for the computer systems to communicate verbally with the users. A voice recognition system, which is an input device, consists of a microphone or telephone that converts human speech into electrical signals. A signal pattern obtained in this manner is transmitted to the computer where it is matched against pre-stored patterns to identify the input. When a close match is found, a word is recognised by the system. The set of pre-stored patterns is known as the vocabulary of the system. To build up this vocabulary, the system has to be trained to recognize the words and phrases that are to be present in the vocabulary. Hence the system is initially operated in a 'training mode' when the user speaks the words and phrases (to be stored in the vocabulary) several times to train the system to recognize his/her particular voice pattern. In this mode, the patterns are created and stored for future matching. Because the accent of different speakers varies, most voice recognition systems, are *speaker-dependent*. However, a system may be trained to recognize the voice of more than one operator in which case a different vocabulary is maintained for each operator.

Although in its infancy, voice recognition systems have found wide application in areas where a person wants to input data to a computerised system in situations where his hands are busy, or his eyes must be fixed on a measuring instrument or some other object. They are also used by security systems for identifying authorised users of the system, by individuals with physical disabilities, and where telephone input is desirable. In fact, voice recognition systems can be used whenever the input is limited and precise.

The potential advantages of voice recognition system are numerous. Their use lowers data entry costs by replacing manual key stroking with verbal instructions. Higher accuracy is also gained when data is entered using voice instead of a keyboard. Obviously, the operator enjoys tremendous freedom of movement with a voice recognition system as compared to a keyboard device. With voice recognition, the operator is free to stand up and move around, entering data with a conveniently located

microphone. Finally, training of operators is an easy task in case of a voice recognition system.

Just as a voice recognition system allows the user to talk to computer, similarly, a voice response system enables a computer to talk to the user. Audio response is an output media that produces verbal responses from the computer system. In a voice response system, all the sounds needed to process the possible enquiries are pre-recorded on a storage medium. Each sound is given a code. When enquiries are received, the computer follows a set of rules to create a reply message in a coded form. This coded message is then transmitted to an audio-response device, which assembles the sounds in the proper sequence and transmits the audio message back to the station requesting the information.

Voice response should only be considered for those applications where there is a low-speed human-machine interaction. The primary business uses of voice response devices are to instruct a person in how to enter data and to output very low volume answers to requests. The banking industry is the largest user of voice response techniques. The most common banking application includes responding to bank account status inquiries over tellers' telephones. Voice response systems are also being used in elevators, by insurance companies in televisions, and by telephone companies. There are also many toys that teach and entertain young children using voice response units. "Touch and Tell" toys that teach basic vocabulary and English have been developed for pre-school children.

## OFFLINE DATA ENTRY DEVICES

*Data entry* is the process of getting data into a form the computer system can understand and use. It will be surprising to note that the cost of data entry alone is around 30 to 50 percent of the total data processing cost of a typical data processing centre.

In general we can consider data entry devices in two broad categories: offline and online. Briefly, online data entry involves devices such as terminals which are connected directly to the computer. On the other hand, offline data entry involves devices through which data is recorded on some media (such as punched cards or tapes) and then entered into the computer later. Since data entry is done by data entry operators and the speed with which a human being can enter data is very very less as compared to the data processing speed of a computer, so in almost 90% of the applications, data entry is done offline. This saves the precious computer time. As the major forms of permanent storage are cards, tapes, and disks, there are offline data entry devices for each type.



### KEY-TO-CARD

It is one of the oldest offline data entry devices. It allows an operator to convert data recorded on paper or other source documents to punched cards. Key punch machines exist for both the 80-column card and the 96-column card. The latest keypunch machines contain electronic circuits and buffers (temporary storage area) that allow them to temporarily store (remember) data, allowing the keypunch operator to edit and correct errors before the data is punched on cards.

### KEY-TO-TAPE

A key-to-tape device, also known as *magnetic tape encoder* is designed to record keyed data directly onto magnetic tape. An operator, keys the source data electronically using a typewriter-like keyboard. The data is stored temporarily by the device and typically displayed on a CRT for visual checking of the correctness of entered data before being transferred to magnetic tape. Magnetic tape encoders are available for recording data magnetically on reels, on cassettes and on tape cartridges. The magnetic tape reels produced by key-to-tape systems are in a computer-compatible format for subsequent direct data input into a computer. However, data on cartridges and cassettes often are transferred to higher-speed media, such as full sized reel of magnetic tape or magnetic disk, for efficient data transfer to the computer.

### KEY-TO-FLOPPY

These data entry machines have been introduced as low-cost data recording systems. They are used to store data directly on flexible disks, called diskettes or floppies that are inexpensive and reusable. A key-to-floppy data entry system consists of a typewriter-like keyboard with additional control keys and a display screen which displays each character keyed in. A data entry operator records data from the source document on the floppy by depressing appropriate keys on the keyboard. The keyed data is held in a buffer (temporary storage area) in a coded form and is also displayed on the display screen. While entering, the operator can verify the correctness of the entered data by seeing the characters displayed on the display screen. If an error is made in entering a data item, the operator can backspace and re-enter it. When the data is found to be correct and the operator wants to store it on the floppy, he presses a control key which causes the data to be stored on the floppy from the buffer.

The same machine can also be used to verify the correctness of the data which has already been stored on a floppy. For this, the machine is operated in a *verify mode*.

and another operator keys in the same data from the source document with the floppy, having the stored data, inserted in the slot. In this mode, the data keyed in and stored in a buffer is compared with the data in the floppy by reading the data from the floppy. If the two data do not agree, an audible and visual signal alerts the operator. The operator re-enters the data which replaces the incorrect data.

### KEY-TO-DISK

Key-to-disk devices are used as data recording stations in systems where data from different points has to be recorded for processing at one point. Typically, from 8 to 64 keyboard stations are linked to a small special-purpose computer to record source data on a disk. Each station has a keyboard and a CRT display. Information entered via the keyboard is displayed on the CRT to allow a visual check. The small special-purpose computer is required to control the input from the various stations enabling the data to be held temporarily for verification and editing before allocation to the disk storage. Thus, these data entry systems are able to correct data before storing it on a magnetic disk and before its entry into the main computer system.

## QUESTIONS

1. Why are I/O devices necessary for a computer system ?
2. What are peripheral devices ? Why are they so called ?
3. What is an I/O processor ? Give two synonyms for it.
4. Why are I/O devices very slow compared to the speed of primary storage and CPU ?
5. Why are punched hole I/O devices so called ?
6. Which coding system is used to represent data on an 80-column card ? Why is this coding system so called ?
7. Differentiate between a zone punch and a numeric punch position for 80-column cards. How will the letter 'E' be represented on an 80-column card using these punch positions ?
8. Explain the different methods used to code data in 80-column and 96-column punched cards.



9. Explain the working of a card punch machine and a card verify machine.
10. What is a card reader ? Briefly describe the way it works.
11. Why are card readers designed to read each card twice ?
12. A 96-column card reader operates at a speed of 1000 cards per minute. Find out its data transfer rate in characters per second.
13. Give the advantages and limitations of cards as compared to tapes and disks as a data storage media.
14. Briefly describe the functioning of a paper tape punch machine and a paper tape reader.
15. Discuss the advantages and limitations of paper tapes as compared to cards as a data storage media.
16. A paper-tape reader reads at the rate of 1200 frames (characters) per second. What card reader speed is this equivalent to in cards per minute for an 80-column card reader ?
17. What is a printer ? What are the three types of printers ?
18. Differentiate between impact and nonimpact printers. Give one example of each.
19. What type of printer is most commonly used with microcomputer systems ?
20. Write short notes on :
  - (a) letter quality printer
  - (b) dot-matrix printer
  - (c) ink-jet printer.
21. What are line printers ? Why are they so called ?
22. What is the difference between a drum printer and a chain printer ?
23. What is a page printer ? Why is it so called ? In what type of computer installations is it used ?
24. What are keyboard devices ? List out some of their characteristics.
25. What is the most popular I/O device in use today ?
26. Explain the functioning of a video display terminal.
27. What is an alphanumeric display ? A graphic display ?
28. How is a light pen used ? Is it an input or output device ?
29. Discuss some of the uses of a graphic display terminal. How can pictures displayed on a terminal screen be permanently preserved ?
30. What is a teleprinter terminal ? How may teleprinters be used ?
31. What is the purpose of a POS terminal ? What advantages are there in capturing data at the transaction source ?
32. Differentiate between a dumb, smart, and an intelligent terminal.
33. List out some of the characteristics of scanners.
34. What is an optical scanner ?
35. What are the three most common types of optical scanners ? What type of data does each scan ?
36. What is the difference between an optical mark reader and an optical character reader ?
37. What is MICR ? What industry is the primary user of MICR ?
38. What are the shortcomings of MICR ?
39. What is the Universal Product Code ? How is it used ?
40. What is COM ? When is it used ?
41. What are the various advantages and limitations of a COM device ?
42. What is a digitizer ?
43. What is a plotter ? What is its principal use ?



44. What is the difference between a graphic terminal and a plotter ?
45. Explain how a voice recognition system works.
46. Under what conditions should a voice response unit be considered ?
47. Voice recognition systems may be speaker-dependent or speaker-independent. How do these two systems differ ?
48. What is an offline data entry system ? In most applications, why are they preferred to an online data entry system ?
49. What is the difference between an input device and an offline data entry device ?
50. How are errors detected in input data ?
51. Explain the functioning of various offline data entry devices.
52. Give the full form of the following abbreviations: VDT, CRT, IOP, CAD, POS, OMR, OCR, UPC, MICR, and COM.
53. Assume that a CPU can add two numbers on an average in 2 microseconds (1 microsecond =  $10^{-6}$  seconds) and a card reader can read cards at a speed of 800 cards per minute. Find the number of additions that can be performed in the time required to read a card. Use this example to support the statement that there exists a considerable mismatch in speed between the peripheral devices and the CPU.
54. Indicate whether the following peripheral devices can be used for input, output, or both :
  - (a) card reader
  - (b) line printer
  - (c) magnetic tape
  - (d) magnetic disk
  - (e) visual display terminal
  - (f) light pen visual display terminal
  - (g) teleprinter terminal
  - (h) POS terminal
  - (i) OCR device
  - (j) COM device
  - (k) digitizer
  - (l) plotter
55. How are graphs and pictures fed to a computer ?



## CHAPTER 10

# 10. COMPUTER SOFTWARE

The terms hardware and software are frequently mentioned in connection with computers. *Hardware* is the jargon term given to the machinery itself and to the various individual pieces of equipment. It refers to the physical devices of a computer system. Thus, the input, storage, processing, control, and output devices are hardware. When the hardware is linked together to form an effective working unit we have a computer installation. In fact, what we have described so far in the previous chapters is actually the hardware of computer systems. The term 'software' will be introduced in this chapter and will be discussed at length in the next few chapters.

### WHAT IS SOFTWARE

It is important to note that a computer cannot do anything on its own. It must be instructed to do a desired job. Hence it is necessary to specify a sequence of instructions that a computer must perform to solve a problem. Such a sequence of instructions written in a language that can be understood by a computer is called a *computer program*. It is the program that controls the activity of processing by the computer and the computer performs precisely what the program wants it to do. The

term *software* refers to the set of computer programs, procedures, and associated documents (flowcharts, manuals, etc.) that describe the programs and how they are to be used. To be precise, software means a collection of programs whose objective is to enhance the capabilities of the hardware machine.

### RELATIONSHIP BETWEEN HARDWARE AND SOFTWARE

In order for a computer to produce useful output, the hardware and software must work together. Thus, there is a special relationship between hardware and software. Both are complementary to each other. Nothing useful can be done with the computer hardware on its own and software cannot be utilised without supporting hardware.

To take an analogy, a cassette player and the cassettes purchased from market are hardware. However, the songs recorded on the cassettes are its software. To listen to a particular song, first of all that song has to be recorded on one of the cassettes which should then be mounted on the cassette player and played. Similarly, to get a particular job done by a computer, the relevant software



should be loaded in the hardware before processing starts. It is immediately evident that hardware is necessary but software is vital. Another important point brought out by this analogy is that software production is difficult and expensive. Hardware is a one-time expense whereas software is a continuing expense. It may also be noted that different softwares can be loaded and run on the same hardware to perform different types of jobs just as different songs can be recorded one by one on the same cassette and played using the cassette player.

## TYPES OF SOFTWARE

Computer software is normally classified into two broad categories : application software and systems software. *Application software*, also known as an *application package*, is a set of one or more programs designed to carry out operations for a specified application. For example, a payroll package produces payslips as the major output and an application package for processing examination results produces marksheets as the major output along with some other statistical reports. Similarly, a program written by a scientist to solve his particular research problem is also an application software. The programs that constitute an application package are known as *application programs* and the person who prepares application programs is known as an *application programmer*.

Traditionally, application packages were limited to the general purpose functions of inventory control, production scheduling, general ledger, general accounting and payroll because these were the major application areas for which a company acquired a computer. Because there is such a great demand for these functions, nowadays packaged software supporting them is available on almost every kind of computer hardware. However, the software industry has expanded to serve many specific markets. Specific purpose application packages have also been developed for specialized areas, such as banking, hospital administration, insurance, publishing, manufacturing, science and engineering, etc. In fact, many software companies have started to offer computer systems to go along with their software as turnkey systems for specific industry needs. Development of specific purpose application packages has expanded along with the increased use of computers in non-financial operations.

*Systems software*, also known as a *systems package*, is a set of one or more programs, designed to control the operation of a computer system. These programs do not solve specific problems. They are general programs written to assist humans in the use of the computer system by performing tasks, such as controlling all of the operations,

required to move data into and out of a computer and all the steps in executing an application program. In general, system packages support the running of other software; communicate with peripheral devices (printers, card readers, disk and tape devices, etc.); support the development of other types of software; and monitor the use of various hardware resources (memory, peripherals, CPU, etc.). Thus, systems software makes the operation of the computer system more effective and efficient. The programs included in a systems software are called *systems programs* and the person who prepares systems software is referred to as a *systems programmer*.

Systems packages offer several advantages and conveniences to application programmers and computer users in general. Good systems software allows application packages to be run on the computer with less time and effort. Without systems software, application packages could not be run on the computer system. However, the production of systems software is a complex task that requires extensive knowledge and considerable specialized training in computer science. Systems programmers, who prepare systems software, are highly trained computer specialists and important members of the computer architectural team. Because of its technical complexity, systems software is rarely developed in-house. They are normally developed and distributed by the computer manufacturers. The customer who buys or leases a computer system would usually receive, in addition to the hardware, some software needed for the effective operation of his computer. The systems software is an indispensable part of a total computer system. Its function is to compensate for the differences that exist between the user needs and the capabilities of the hardware. A computer without some kind of systems software would be very ineffective and most likely impossible to operate.

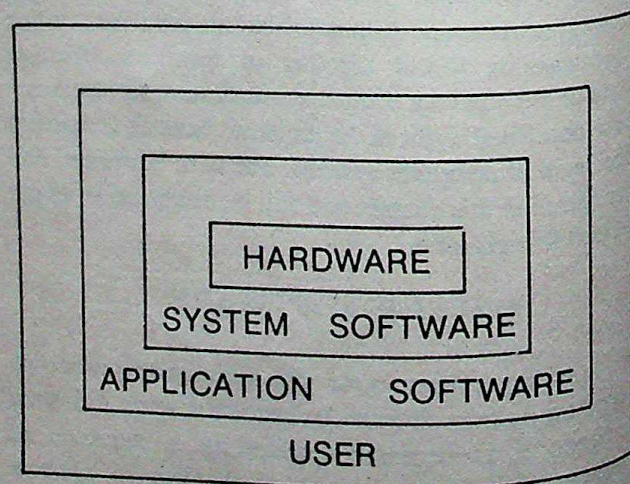


Figure 10.1. Relationship between hardware, software and the user of a computer system.



The relationship between hardware, software and the user of a computer system is shown in Figure 10.1. At the centre of any computer system is hardware and surrounding the hardware is systems software. Finally, the user interacts with application software.

## ACQUIRING SOFTWARE

Computer software has evolved from an item of minor economic concern with first generation digital computers to one of major concern in today's computing systems. In the early days of computers, more than 80% of the total computer cost was devoted to hardware, but today more than 80% of the cost is for software. At one time, application and systems software were included in the purchase price of the computer. Today, however, software is usually not included in the purchase price of the computer. For most computer manufacturers the purchase price of a computer includes only the hardware along with a minimum of systems software. The customer normally has to pay extra charges for additional systems software and application software that he may wish to purchase.

Basically, there are two ways of obtaining software. One can either buy an existing package, or he can develop his own package. There are many financial options if one decides to buy a software package. It can be purchased, leased or rented, or obtained through a lease-purchase agreement. Software can be purchased from the computer manufacturers, from a software company or software house that specializes in developing and selling software. Software can also be obtained from user groups, which are groups of people or organizations that use the same computer and share software. On placement of advance order, many computer manufacturers also supply specialized application packages that are developed only for a particular user for his specific need. These packages are normally supplied along with the computer system, which is normally referred to as a *turnkey system*. If the software is to be developed in-house, that is, if the user decides to develop his own package, then some computer professionals are required to plan, develop, test, implement, and maintain the computer programs. However, development and maintenance of software is not an easy task. It is a very costly and time consuming affair. Hence, thorough analysis should be done and considerable thought should be given before one arrives at a decision of either buying or developing a software package.

## FIRMWARE

Computer software in conventional systems is supplied on storage media like floppies, tapes, disks, etc.

However, with the advancement in technology and the reduction in hardware cost, today software is also being made available by many computer manufacturers on read-only memory (ROM) chips. These ROM chips can be easily plugged into the computer system and they form a part of the hardware. Such programs (software) made available on hardware are known as *firmware*. Firmware often refers to a sequence of instructions (software) that is substituted for hardware. For example, in an instance where cost is more important than performance speed, the computer system architect might decide not to use special electronic circuits (hardware) to multiply two numbers, but instead write instructions (software) to cause the machine to accomplish the same function by repeated use of circuits already designed to perform addition. This software will be stored in a ROM chip of the computer system and will be executed (used) whenever the computer has to multiply two numbers. Hence, this software will be known as firmware. To be precise, firmware is software substituted for hardware and stored in read-only memory.

Initially, only systems software was supplied in the form of firmware. But today, even application programs are being supplied in firmware form. Dedicated applications are also programmed in this fashion and available in firmware. Because of the rapid improvements in memory technology, firmware is frequently a cost-effective alternative to wired electronic circuits, and its use in computer design will increase. It is expected that in the near future, firmware will make possible the cost-effective production of smart machines of all types.

## QUESTIONS

1. Define the terms hardware and software.
2. What is a computer program ?
3. Hardware and software of a computer system are like two sides of a coin. Discuss.
4. Give an analogy to bring out the relationship between hardware and software of a computer system.
5. Hardware is a one-time expense whereas software is a continuing expense. Elaborate.
6. How many types of software are there ? Give two examples of each.



7. Define and distinguish between application software and systems software.
8. What type of qualification is desirable for a systems programmer and why ?
9. Highlight the importance of systems software for a computer system.
10. How does a normal user interact with the hardware of a computer system ? Describe the various in-between layers.
11. What are the different ways of acquiring software ?
12. What is a turnkey system ?
13. What is firmware and what is its importance to the computer system architect ?
14. Why is firmware gaining popularity ?
15. Differentiate between hardware, software, and firmware.



## CHAPTER 11

# 11. PLANNING THE COMPUTER PROGRAM

In Chapter 10, computer programs have been described as the software of the computer system. We are now at the point in the system life cycle where programs are to be written. But before writing a computer program, the programmer must be clear about the processing steps to be performed by the computer. Thus, to produce an effective computer program, one must first plan the logic (the various steps) of the program. If one attempts to plan the logic approach and write the program at the same time, he will likely become so involved with the required instructions that program logic will suffer. Hence, before we learn how to write a program (next chapter), we will first learn how to plan the logic of a computer program in this chapter.

Suppose you are asked by your teacher to solve an arithmetic problem and you are not familiar with the steps involved in solving that problem. In such a situation, you will not be able to solve the problem. The same principle applies to writing computer programs also. A programmer cannot write the instructions to be followed by a computer unless the programmer knows how to solve the problem manually.

Suppose you know the steps to be followed for solving the given problem but while solving the problem, you forget to apply some of the steps or you apply the calculation steps in the wrong sequence. Obviously, you will get a wrong answer. Similarly, while writing a computer program, if the programmer leaves out some of the instructions for the computer or writes the instructions in the wrong sequence, then the computer will calculate a wrong answer. Thus, to produce an effective computer program, it is necessary that the programmer writes each

### PURPOSE OF PROGRAM PLANNING

Suppose you are asked by your teacher to solve an



and every instruction in the proper sequence. However, the instruction sequence (logic) of a computer program can be very complex. Hence, in order to ensure that the program instructions are appropriate for the problem and are in the correct sequence, programs must be planned before they are written.

## ALGORITHM

The term algorithm may be formally defined as a sequence of instructions designed in such a way that if the instructions are executed in the specified sequence, the desired results will be obtained. The instructions, however, should be precise and unambiguous and the result should be obtained after a finite number of executional steps. The latter condition actually states that an algorithm must terminate and should not repeat one or more instructions infinitely. In other words, the algorithm represents the logic of the processing to be performed. However, in order to qualify as an algorithm, a sequence of instructions must possess the following characteristics :

1. Each and every instruction should be precise and unambiguous.
2. Each instruction should be such that it can be performed in a finite time.
3. One or more instructions should not be repeated infinitely. This ensures that the algorithm will ultimately terminate.
4. After performing the instructions, that is after the algorithm terminates, the desired results must be obtained.

To gain insight into algorithms, let us consider a simple example.

**Example 11.1.** There are 50 students in a class who appeared in their final examination. Their marksheets have been given to you. Write an algorithm to calculate and print the total number of students who passed in first division.

**Algorithm :**

**Step 1 :** Initialize TOTAL FIRST DIVISION and TOTAL MARK SHEETS CHECKED to zero.

**Step 2 :** Take the marksheet of the next student.

**Step 3 :** Check the division column of the marksheet to see if it is I : if no, go to step 5.

**Step 4 :** Add 1 to TOTAL FIRST DIVISION.

**Step 5 :** Add 1 to TOTAL MARKSHEET CHECKED.

**Step 6 :** Is TOTAL MARKSHEET CHECKED = 50 : if no, go to step 2.

**Step 7 :** Print TOTAL FIRST DIVISION.

**Step 8 :** Stop.

It must be clear to the readers from this example that even for very simple problems, the development of algorithms is not so simple as it might initially appear and requires some thinking. It may also be noted from the given example that in order to solve a given problem, each and every instruction must be strictly carried out in a particular sequence. It is this fact which a beginner to problem solving by computers finds difficult to appreciate.

There are various ways in which an algorithm can be expressed. When an algorithm is expressed in a programming language, it becomes a program. Thus, a program is an algorithm although the reverse is not true. Besides represented as programs, algorithms are often expressed in the form of flowcharts which is discussed below.

## FLOWCHARTS

A flowchart is a pictorial representation of an algorithm that uses boxes of different shapes to denote different types of instructions. The actual instructions are written within these boxes using clear and concise statements. These boxes are connected by solid lines having arrow marks to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed.

Normally, an algorithm is first represented in the form of a flowchart and the flowchart is then expressed in some programming language to prepare a computer program. The main advantage of this two step approach to program writing is that while drawing a flowchart one is not concerned with the details of the elements of a programming language. Hence, he can fully concentrate on the logic of the procedure. Moreover, since a flowchart shows the flow of operations in pictorial form, any error in the logic of the procedure can be detected more easily than in the case of a program. Once the flowchart is ready,



programmer can forget about the logic and can concentrate only on coding the operations in each box of the flowchart in terms of the statements of the programming language. This will normally ensure an error-free program.

A flowchart, therefore, is a picture of the logic to be included in the computer program. It is simply a method of assisting the programmer to lay out, in a visual, two-dimensional format, ideas on how to organise a sequence of steps necessary to solve a problem by a computer. It is basically the plan to be followed when the program is written. It acts like a road map for a programmer and guides him how to go from the starting point to the final point while writing a computer program.

Experienced programmers sometimes write programs without drawing the flowchart. However, for a beginner it is recommended that a flowchart be drawn first in order to reduce the number of errors and omissions in the program. Moreover, it is a good practice to have a flowchart along with a computer program because a flowchart is very helpful during the testing of the program as well as while incorporating further modifications in the program.

### FLOWCHART SYMBOLS

We have seen that a flowchart uses boxes of different shapes to denote different types of instructions. The communication of program logic through flowcharts is made easier through the use of symbols that have standardized meanings. For example, a diamond always means a decision. Only a few symbols are needed to indicate the necessary operations in a flowchart. These symbols have been standardised by the American National Standards Institute (ANSI). These symbols are shown in Figure 11.1 and their functions are discussed below.

**Terminal.** The terminal symbol, as the name implies, is used to indicate the beginning (START), ending (STOP), and pauses (HALT) in the program logic flow. It is the first symbol and the last symbol in the program logic. In addition, if the program logic calls for a pause in the program, that also is indicated with a terminal symbol. A pause is normally used in the program logic under some error conditions or if forms had to be changed in the computer's line printer during the processing of that program.

**Input/Output.** The input/output symbol is used to denote any function of an input/output device in the program. If there is a program instruction to input data from a disk, tape, card reader, terminal, or any other type of input device, that step will be indicated in the flowchart with an

input/output symbol. Similarly, all output instructions, whether it is output on a printer, magnetic tape, magnetic disk, terminal screen, or any output device, are indicated in the flowchart with an input/output symbol.

**Processing.** A processing symbol is used in a flowchart to represent arithmetic and data movement instructions. Thus, all arithmetic processes of adding, subtracting, multiplying and dividing are shown by a processing symbol. The logical process of moving data from one location of the main memory to another is also denoted by this symbol. When more than one arithmetic and data movement instructions are to be executed consecutively, they are normally placed in the same processing box and they are assumed to be executed in the order of their appearance.

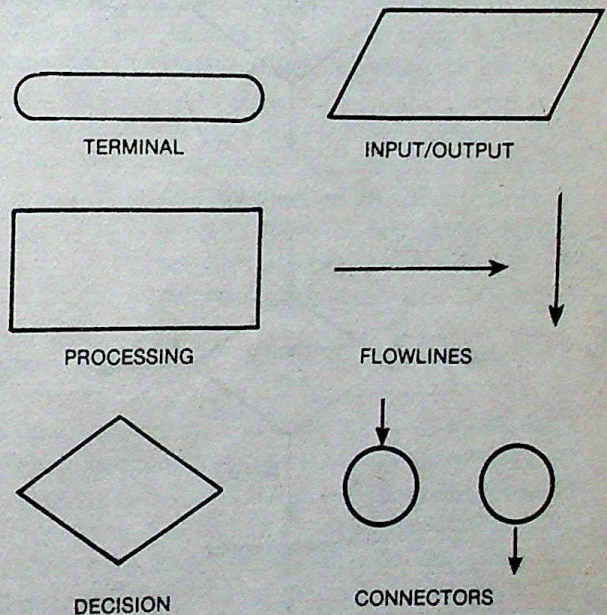


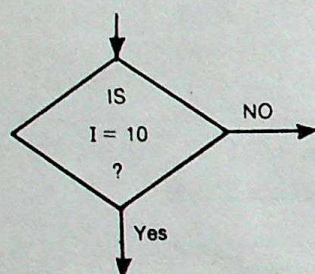
Figure 11.1. Various flowchart symbols.

**Flowlines.** Flowlines with arrowheads are used to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed. The normal flow of flowchart is from top to bottom and left to right. Arrowheads are required only when the normal top to bottom flow is not to be followed. However, as a good practice and in order to avoid ambiguity, flowlines are usually drawn with an arrowhead at the point of entry to a

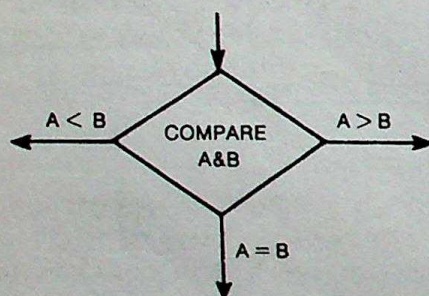


symbol. Good practice also dictates that flowlines should not cross each other and that such intersections should be avoided whenever possible.

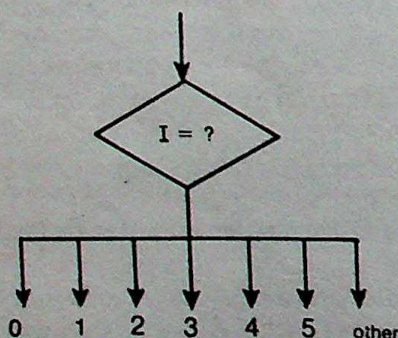
**Decision.** The decision symbol is used in a flowchart to indicate a point at which a decision has to be made and a branch to one of two or more alternative points is possible. Figure 11.2 shows three different ways in which a decision symbol can be used. It may be noted from these examples that the criterion for making the decision should be indicated clearly within the decision box. Moreover, the condition upon which each of the possible exit paths will be executed should be identified and all the possible paths should be accounted for. During execution, the appropriate path is followed depending upon the result of the decision.



(a) Two-way branch



(b) Three-way branch



(c) Multiple-way branch

Figure 11.2. Examples of decision symbols.

**Connector.** If a flowchart becomes very long, the flowlines start criss-crossing at many places that causes confusion and reduces understandability of the flowchart. Moreover, there are instances when a flowchart becomes too long to fit in a single page and the use of flowlines becomes impossible. Thus, whenever a flowchart becomes complex enough that the number and direction of flowlines is confusing or it spreads over more than one page, it is useful to utilize the connector symbol as a substitute for flowlines. This symbol represents an entry from, or an exit to another part of the flowchart. A connector symbol is represented by a circle and a letter or digit is placed within the circle to indicate the link. A pair of identically labeled connector symbols is commonly used to indicate a continued flow when the use of a line is confusing. So two connectors with identical labels serve the same function as a long flowline. That is, they show an exit to some other chart section, or they indicate an entry from another part of the chart. How is it possible to determine if a connector is used as an entry or an exit point? It is very simple: if an arrow enters but does not leave a connector, it is an exit point and program control is transferred to the identically labeled connector that does have an outlet. It may be noted that connectors do not represent any operation and their use in a flowchart is only for the sake of convenience and clarity.

## SAMPLE FLOWCHARTS

A flowchart should be drawn using the symbols mentioned above. To describe an algorithm in the form of a flowchart is not very difficult. What is required is some common sense and a little practice. The art of flowcharting is introduced below with the help of some simple examples.

### Example 11.2.

A student appears in an examination that consists of total 10 subjects, each subject having maximum marks of 100. The roll number of the student, his name, and the marks obtained by him in various subjects is supplied as input data. Such a collection of related data items that is treated as a unit is known as a *record*. Draw a flowchart for the algorithm to calculate the percentage marks obtained by the student in this examination and then to print it along with his roll number and name.

### Solution :

The flowchart for the algorithm of this problem is shown in Figure 11.3. The first symbol is a terminal labeled START. It shows that this is the starting point or beginning of our flowchart logic. It does not mean that the computer is



to be turned on or that anyone is to press a start button. The second symbol is an I/O symbol that is labeled specifically to show that this step is READ INPUT DATA. This step will input the roll number, name, and the marks obtained by the student from an input device into the main storage of the computer system. The third symbol is a processing symbol which is suitably labeled to indicate that at this step, the computer will add the marks obtained by the student in various subjects and then store the sum in a memory location which has been given the name TOTAL. The fourth symbol is again a processing symbol. The label inside it clearly indicates that the percentage marks obtained by the student is calculated at this stage by dividing TOTAL by 10 and the result is stored in a memory location which has been given the name PERCENTAGE. The fifth symbol is an I/O symbol and is labeled WRITE OUTPUT DATA. This logical

such as the roll number, name, and the marks or percentage being inputted or outputted or the specific positions being used are not a part of the logical steps of inputting or outputting. This information already appears in the system design documents and will be included in the computer program as input and output descriptions. The sixth symbol is a terminal symbol labeled STOP. This symbol indicates the conclusion of our logic - that is, the conclusion of the computer program. The various symbols used in the flowchart are connected by directed flowlines to indicate the sequence in which the instructions are to be executed.

The logic depicted in Figure 11.3 therefore, will read the students's record, calculate the percentage marks obtained by him, print one line, and then stop. One would certainly not like to use a computer to solve a trivial problem such as this. However, if we have to compute the percentage marks obtained by several students in the same examination then we may like to take the help of a computer. The next example illustrates how to do this.

**Example 11.3.** 50 students of a class appear in the examination of Example 11.2. Draw a flowchart for the algorithm to calculate and print the percentage marks obtained by each student along with his roll number and name.

**Solution :**

Since all the students have appeared in the same examination, so the process of calculation and printing the percentage marks obtained by each student will basically remain the same. The same process of reading the input data, adding the marks of all subjects, calculating the percentage, and then writing the output data has to be repeated for all the 50 students. Hence, an easy solution that comes to ones mind for this problem is to repeat the intermediate four symbols of Figure 11.3 fifty times. However if that is done, a total of 202 ( $50 \times 4 + 2$ ) flowchart symbols will have to be drawn. Obviously this will be a very time consuming and tedious job and hence is not desirable. We will now see how to solve this problem in a simpler way.

In a situation where the same logical steps can be repeated, the flowline symbols are used in a flowchart to indicate the repetitive nature of the logic in the form of a *process loop*. Figure 11.4 illustrates a flowchart with a process loop. Note the arrowhead on the flowline that forms the loop. It points upward indicating that as soon as the WRITE operation is over the control will flow back to the READ operation. Thus, the process loop of Figure 11.4

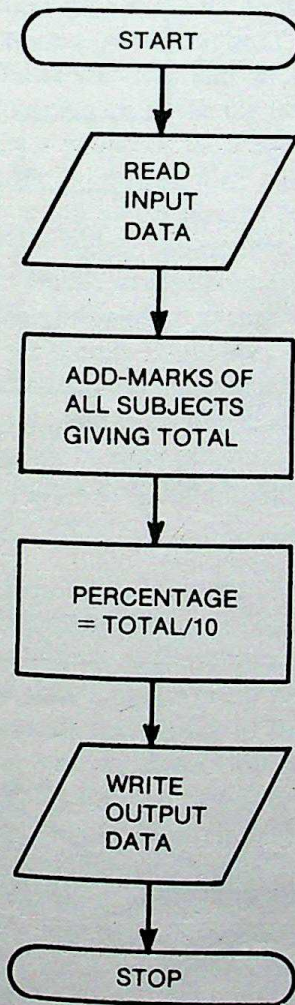


Figure 11.3. Flowchart for Example 11.2.

step in the flowchart indicates that the data desired as output will be outputted on the line printer. Note that details



solves the problem of an exceedingly long flowchart by reusing the same logical steps over and over again. However, the flowchart of Figure 11.4 is incomplete because the process loop has introduced a new problem. The process loop shown does not have a logical ending. It will continue to attempt to repeat those four steps until someone manually cancels the job. This is an example of an *infinite loop* and hence the flowchart of Figure 11.4 does not represent an algorithm because an algorithm must terminate. So we have to find out a way of terminating the algorithm. This is done by the use of a decision symbol.

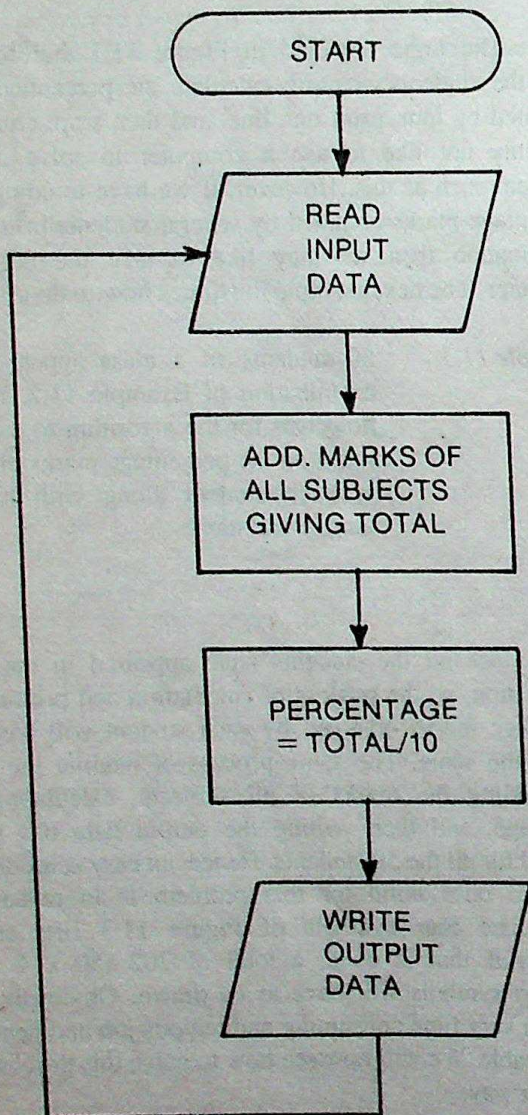


Figure 11.4. Flowchart for the solution of Example 11.3 with an infinite (endless) process loop.

variable COUNT has been introduced which is initialized to zero outside the process loop and is incremented

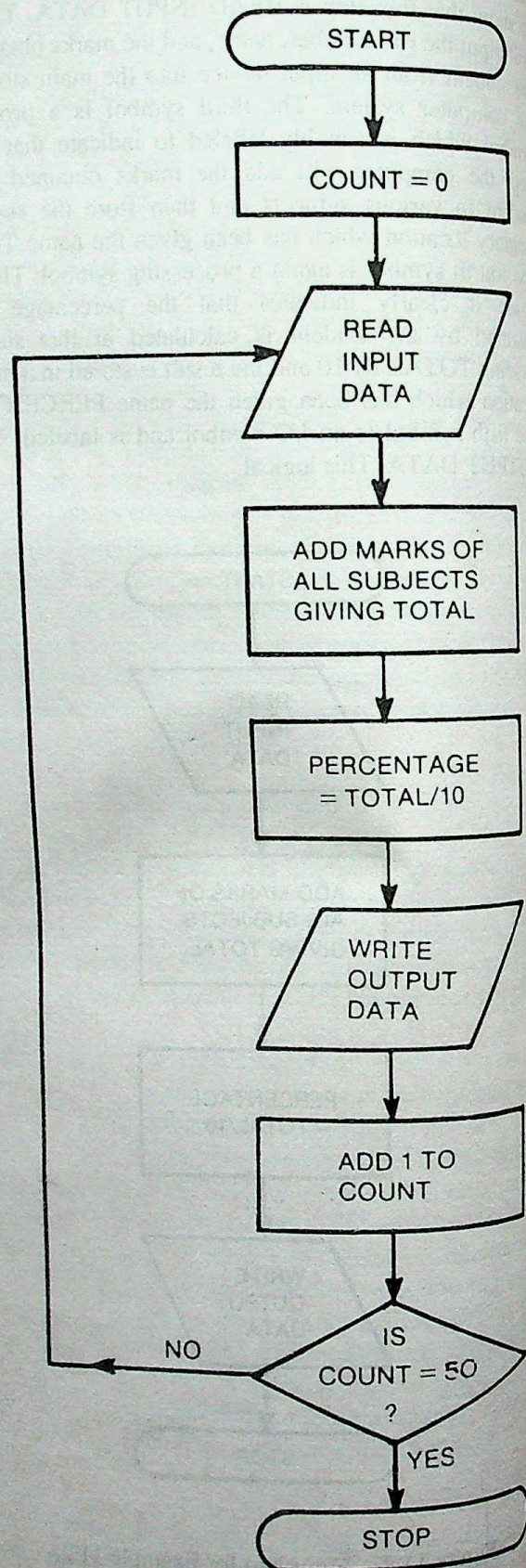


Figure 11.5. Flowchart for the solution of Example 11.3.

Figure 11.5 shows a flowchart which uses a decision step to terminate the algorithm. In this flowchart, another



by 1 after processing the data for each student. Thus, the value of COUNT will always be equal to the number of students whose data has already been processed. At the decision step, the value of COUNT is compared with 50 which is the total number of students who have appeared for the examination. The steps within the process loop are repeated until the value of COUNT becomes equal to 50. As soon as the value of COUNT becomes equal to 50, the instruction at the decision step causes the control to flow out of the loop and the processing stops because a terminal symbol labeled STOP is encountered. Thus, the flowchart of Figure 11.5 is a complete and correct solution to the problem of Example 11.3.

Although the flowchart of Figure 11.5 is a correct solution to the given problem, it suffers from two major drawbacks. The first drawback is that in order to make the decision at the decision step, one must know the exact number of students who appeared in the examination. Suppose the examination of Example 11.2 is a university examination in which the total number of students who appeared for the examination is too big (say more than ten thousand). In such a situation, the counting of the total number of input records (data for each student) becomes a tedious job. Even then, if we do stick on counting the input records manually and supply the number of records to be compared against COUNT in the decision box and suppose while counting we make some error then the logic will not work. If the supplied number is less than the actual number of input records, then the computer will not process the data for last few students. And if the supplied number is more than the actual number of input records, the computer will try to read more records than what is supplied which will cause an error in the logic.

The second drawback is that the flowchart of Figure 11.5 is not a generalized solution to the given problem. Suppose the examination is conducted every year and so we will like to use the same program every year to process the students' data. However, the number of students appearing in the examination may not remain the same every year. This year it may be 50, but next year it can be 55 or 60 or anything. So if the computer program to solve this problem was based on the flowchart of Figure 11.5, the statement concerned with the decision step in that program will have to be changed again and again to supply the exact number of students. This is not a good programming practice. A good program should be general in nature. For example, in this case we should write a program that need not be modified every time even if the total number of students changes.

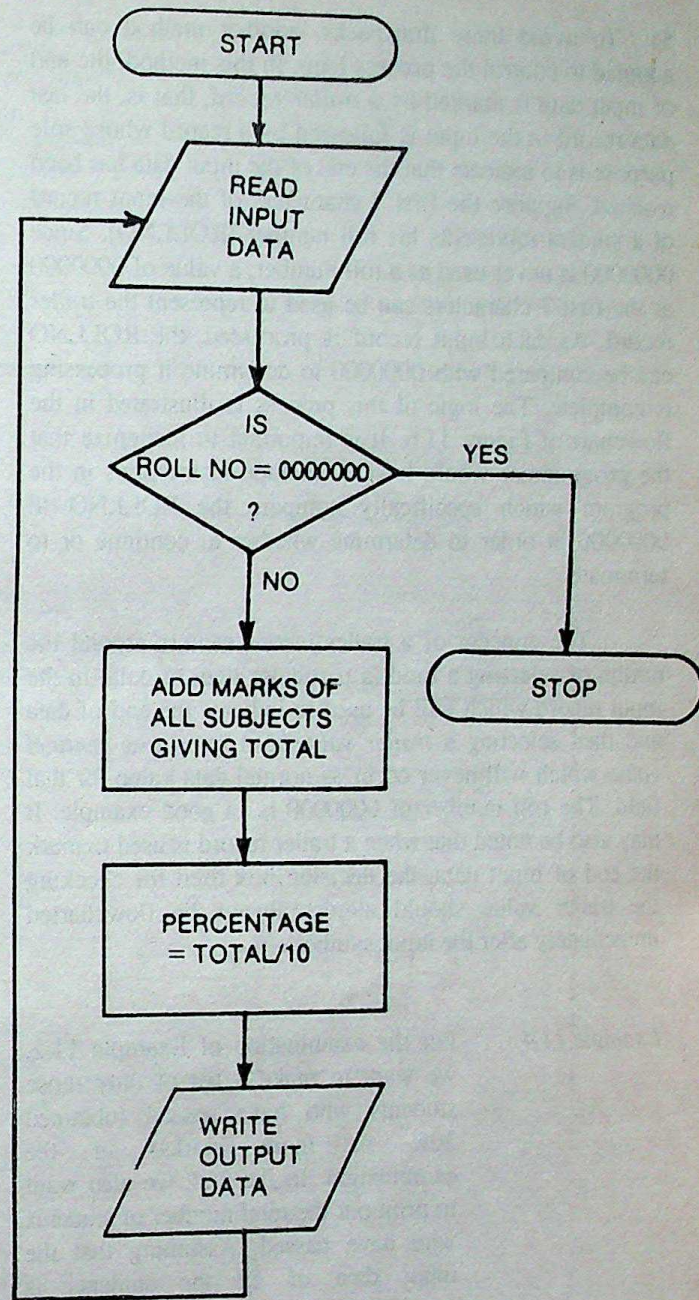


Figure 11.6. Generalised flowchart for the solution of Example 11.3 using the concept of trailer record. Here the process loop is terminated by detecting a special non-data record.

drawbacks because in this flowchart the process loop is being controlled by counting. When the loop has been executed 50 times, the decision will cause execution to proceed to the STOP, thus terminating processing. (The reader should carefully step through the logic to ensure that the loop will be executed exactly 50 times and not 49 or



51.) To avoid these drawbacks, another method can be adopted to control the process loop. In this method, the end of input data is marked by a *trailer record*, that is, the last data record in the input is followed by a record whose sole purpose is to indicate that the end of the input data has been reached. Suppose the first 7 characters of the input record of a student represents his roll number (ROLLNO). Since 0000000 is never used as a roll number, a value of 0000000 as the first 7 characters can be used to represent the trailer record. As each input record is processed, the ROLLNO can be compared with 0000000 to determine if processing is complete. The logic of this process is illustrated in the flowchart of Figure 11.6. It is important to recognize that the programmer would have to include instructions in the program which specifically compare the ROLLNO to 0000000 in order to determine whether to continue or to terminate.

The concept of a trailer record centers around the notion of selecting a field (a particular item of data) in the input record which will be used to indicate the end of data and then selecting a *trailer value* also known as *sentinel value* which will never occur as normal data value for that field. The roll number of 0000000 is a good example. It may also be noted that when a trailer record is used to mark the end of input data, the decision box used for checking the trailer value should almost always be flowcharted immediately after the input symbol.

**Example 11.4.** For the examination of Example 11.2, we want to make a list of only those students who have passed (obtained 30% or more marks) in the examination. In the end, we also want to print out the total number of students who have passed. Assuming that the input data of all the students is terminated by a trailer record that has sentinel value of 9999999 for ROLLNO, draw a flowchart for the algorithm to do the above job.

**Solution :**

The flowchart in Figure 11.7 is a solution to this problem. There are two decision symbols in this flowchart. The first decision symbol checks for a trailer record by comparing ROLLNO against the value 9999999 to determine if processing is complete. The second decision symbol is used to check whether the student has passed or failed by comparing the percentage marks obtained by him against 30. If the student's PERCENTAGE is equal to or more than 30 then he has passed otherwise he has failed.

Note from the flowchart that the operation WRITE OUTPUT DATA is performed only if the student has passed. If he has failed, we directly perform the operation READ INPUT DATA without performing the WRITE operation. This ensures that the output list provided by the computer will contain the details of only those students who have passed in the examination.

Another point to be noted in this flowchart is the use of variable COUNT. This variable has been initialized to zero in the beginning and is incremented by 1 every time the operation WRITE OUTPUT DATA is performed. But we have seen that the operation WRITE OUTPUT DATA is performed only for the students who have passed. Hence, the variable COUNT will be incremented by 1 only in case of students who have passed. Thus, the value of COUNT will always be equal to the number of students whose data has already been processed and who have been identified as passed. Finally, when the trailer record is detected, the operation WRITE COUNT will print out the final value of COUNT that will be equal to the total number of students who have passed the examination.

**Example 11.5.** Suppose the input data of each student for the examination of Example 11.2 also contains information regarding the sex of the candidate in the field named SEXCODE that can have values M (for male) or F (for female). We want to make a list of only those female students who have passed in second division (obtained 45% or more but less than 60% marks). In the end we also want to print out the total number of such students. Assuming that the input data of all the students is terminated by a trailer record that has a sentinel value of Z for SEXCODE draw a flowchart for the algorithm to do the above job.

**Solution :**

The flowchart in Figure 11.8 is a solution to this problem. There are four decision symbols in this flowchart. The first decision symbol checks for a trailer record by comparing SEXCODE against the value Z to determine if processing is complete. The second decision symbol is used to check whether the candidate is female or not by comparing the SEXCODE of that candidate against F. Note that if the SEXCODE is not F, that is, the candidate is not a female, we do not process the data of that student and return back to perform the operation of reading input data. This step ensures that the data of only female students will be taken for further processing.



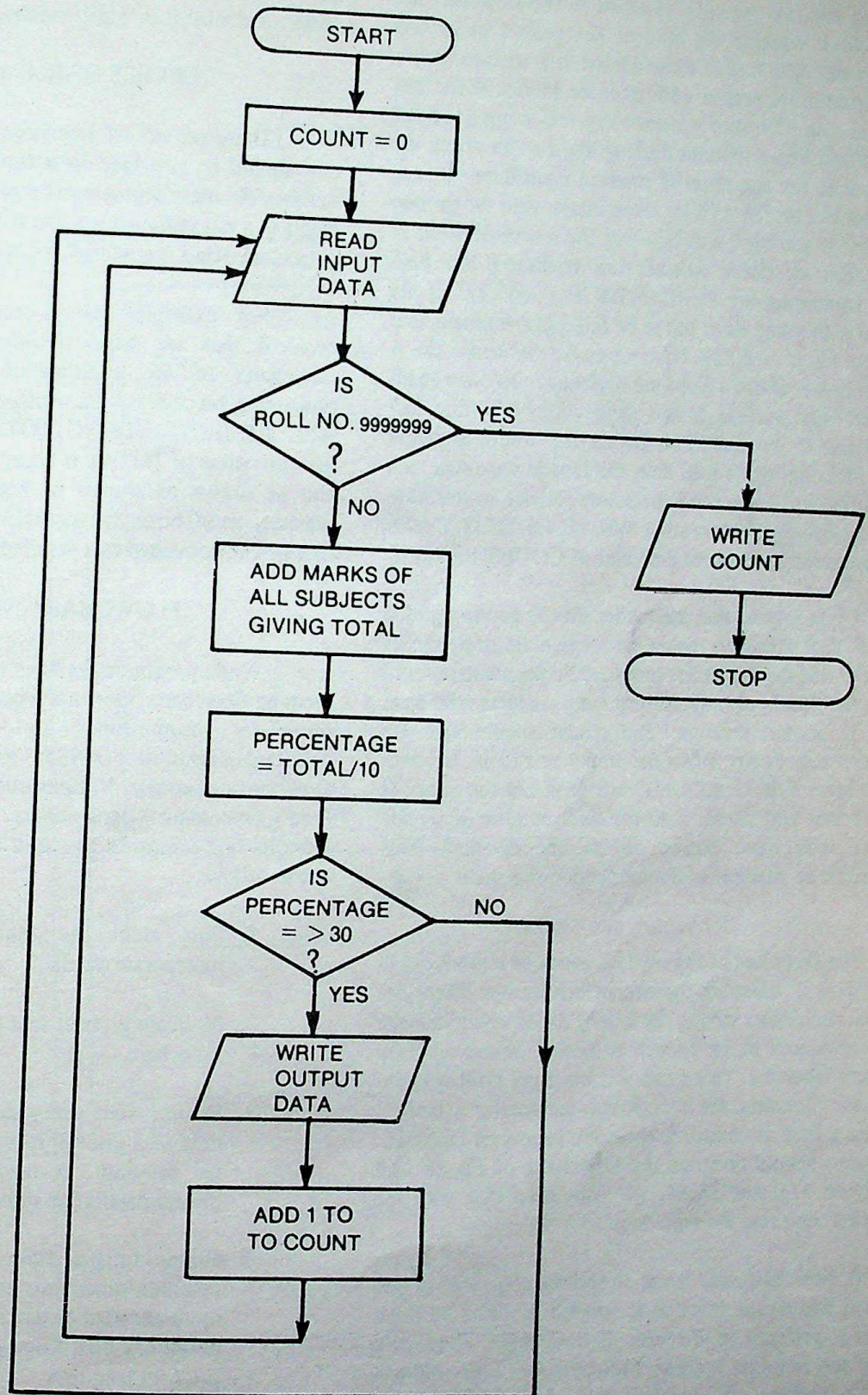


Figure 11.7. Flowchart for the solution of Example 11.4.



The last two decision symbols in the flowchart are used to check whether the student has passed in second division or not. The first of these decisions is used to ensure that the student has scored 45% or more marks. If she has scored less than 45% then it means that she is not a second divisioner and hence without making any further check we return back to the operation of reading input data. In case the student has scored 45% or more marks then we go one step further in the logic and by using the fourth decision in the flowchart we check whether her marks are less than 60% by comparing her PERCENTAGE against 60. If the condition at this step turns out to be false then it means that the student has scored 60% or more marks and hence she is a first divisioner and not a second divisioner. So once again we return back to read a new data without writing any output data. If the condition inside the fourth decision symbol turns out to be true then the female candidate can be classified to be a second divisioner. Hence in this case only we perform the operation WRITE OUTPUT DATA and subsequently increment the value of COUNT by 1.

It is suggested that the reader should go through the logic of this flowchart again and again until he/she is convinced that the output list provided by the computer will contain the details of only those female students who have passed in second division. The reader should also get convinced that finally when the trailer record is detected, the operation WRITE COUNT will print out the value of COUNT that will be equal to the total number of female students who have passed in second division. This flowchart is an example of a multiple-decision chart.

The flowchart of Figure 11.8 has been reproduced in Figure 11.9 to illustrate the use of connectors. There are four exit connectors having the label 1 all of which indicate a flow of control to the flowchart symbol having an entry connector labeled 1. This symbol is the input symbol in the flowchart. Similarly the exit connector having a label 2 indicates a flow of control to the entry connector labeled 2. The reader should compare the flowcharts of Figure 11.8 and Figure 11.9 and should get convinced that both the flowcharts represent the same logic.

A flowchart may seem simple to prepare, but you will find that much practice is needed in order to think through a problem in discrete, logical steps, to assume nothing and to forget nothing. Moreover, not everyone will tackle a problem in exactly the same way and, in consequence, several different flowcharts could be drafted for the same problem. It may also be noted that a completed flowchart is not a complete computer program. It is only an aid to programming. For a given problem, it defines the procedure and the logic involved. From the examples that

have been discussed above, we are in a better position to understand what this 'logic' means.

## LEVELS OF FLOWCHARTS

There are no set standards on the amount of detail that should be provided in a flowchart. A flowchart that outlines the main segments of a program or that shows less detail is a *macroflowchart*. On the other hand, a flowchart with more detail is a *microflowchart*, or detailed flowchart.

For example, let us consider the examination problem that we have already discussed. In all the flowcharts of the examination problem, there is a processing box having the instruction "ADD MARKS OF ALL SUBJECTS GIVING TOTAL". In order to display how the value of TOTAL is computed, a detailed flowchart can be drawn as shown in Figure 11.10. In a similar manner, the I/O boxes for the READ and WRITE operations can also be converted to a detailed flowchart.

## FLOWCHARTING RULES

While programmers have a good deal of freedom in creating flowcharts, there are a number of general rules and guidelines recommended by the American National Standards Institute (ANSI) to help standardize the flowcharting process. Various computer manufacturers and data processing departments usually have similar flowcharting standards. Some of these rules and guidelines are as follows :

1. First chart the main line of logic, then incorporate detail.
2. Maintain a consistent level of detail for a given flowchart.
3. Do not chart every detail or the flowchart will only be a graphic representation, step by step, of the program. A reader who is interested in greater details can refer to the program itself.
4. Words in the flowchart symbols should be common statements and easy to understand. It is recommended to use descriptive titles written in designer's own language rather than in machine oriented language.
5. Be consistent in using names and variables in the flowchart.
6. Go from left to right and top to bottom in constructing flowcharts.



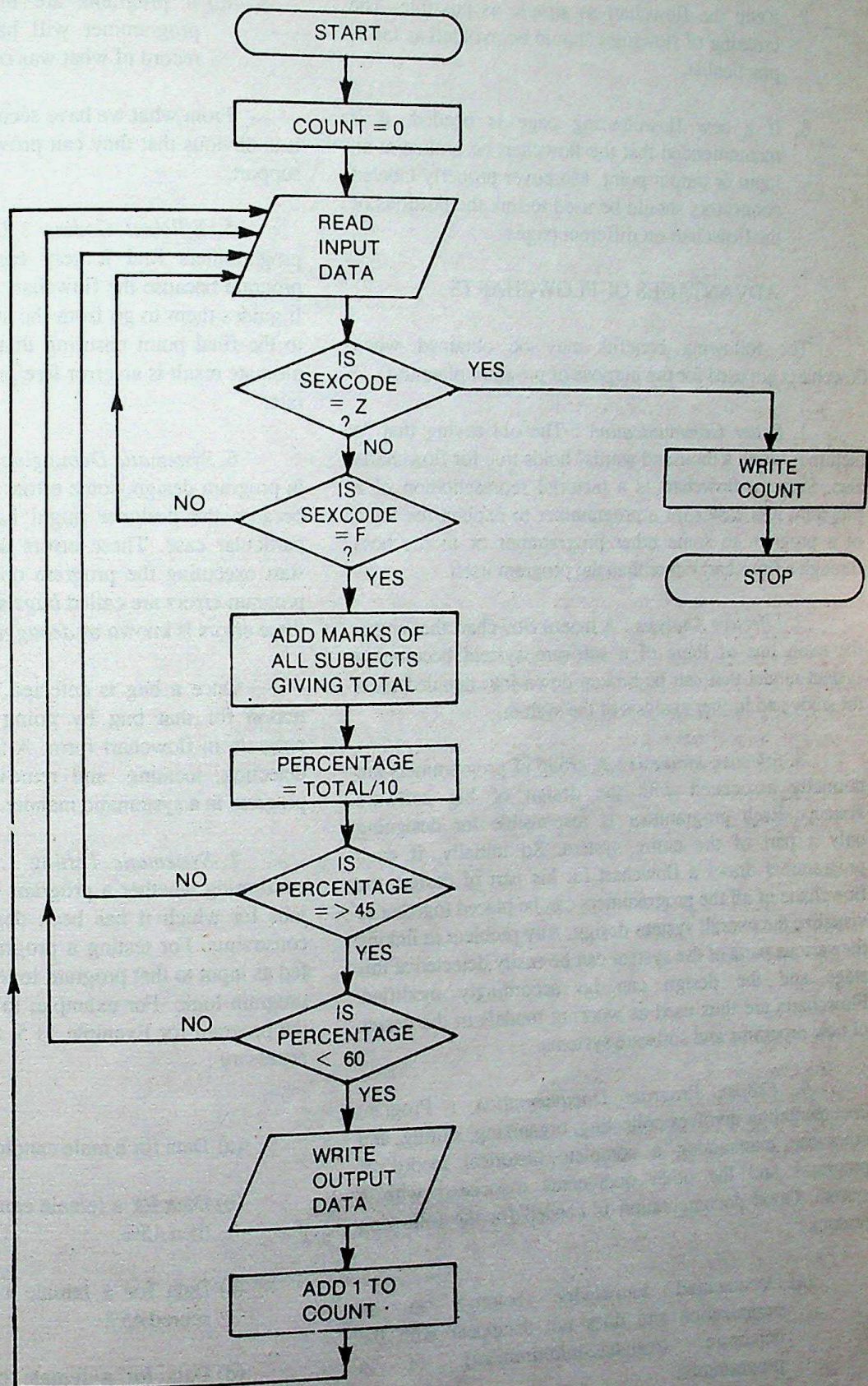


Figure 11.8. Flowchart for the solution of Example 11.5.



7. Keep the flowchart as simple as possible. The crossing of flowlines should be avoided as far as practicable.
8. If a new flowcharting page is needed, it is recommended that the flowchart be broken at an input or output point. Moreover properly labeled connectors should be used to link the portions of the flowchart on different pages.

### ADVANTAGES OF FLOWCHARTS

The following benefits may be obtained when flowcharts are used for the purpose of program planning.

1. *Better Communication* : The old saying that "a picture is worth a thousand words" holds true for flowcharts also. Since, a flowchart is a pictorial representation of a program, it is easier for a programmer to explain the logic of a program to some other programmer or to his boss through a flowchart rather than the program itself.

2. *Effective Analysis* : A macro flowchart that charts the main line of logic of a software system becomes a system model that can be broken down into detailed parts for study and further analysis of the system.

3. *Effective synthesis* : A group of programmers are normally associated with the design of big software systems. Each programmer is responsible for designing only a part of the entire system. So initially, if each programmer draws a flowchart for his part of design, the flowcharts of all the programmers can be placed together to visualize the overall system design. Any problem in linking the various parts of the system can be easily detected at this stage and the design can be accordingly modified. Flowcharts are thus used as working models in the design of new programs and software systems.

4. *Proper Program Documentation* : Program documentation involves collecting, organizing, storing, and otherwise maintaining a complete historical record of programs and the other documents associated with a system. Good documentation is needed for the following reasons :

(a) Documented knowledge belongs to an organization and does not disappear with the departure (resignation/retirement) of a programmer.

(b) If projects are postponed, documented work will not have to be duplicated.

(c) If programs are modified in the future, the programmer will have a more understandable record of what was originally done.

From what we have seen of the nature of flowcharts, it is obvious that they can provide valuable documentation support.

5. *Efficient Coding* : Once a flowchart is ready, programmers find it very easy to write the concerned program because the flowchart acts as a roadmap for them. It guides them to go from the starting point of the program to the final point ensuring that no steps are omitted. The ultimate result is an error free program developed at a faster rate.

6. *Systematic Debugging* : Even after taking full care in program design, some errors may remain in the program because the designer might have never thought about a particular case. These errors are detected only when we start executing the program on a computer. Such type of program errors are called *bugs* and the process of removing these errors is known as *debugging*.

Once a bug is detected, it is easier to find out the reason for that bug by going through the logic of the program in flowchart form. A flowchart is very helpful in detecting, locating, and removing mistakes (bugs) in a program in a systematic manner.

7. *Systematic Testing* : Testing is the process of confirming whether a program will successfully do all the jobs for which it has been designed under the specified constraints. For testing a program, different set of data is fed as input to that program to test the different paths in the program logic. For example, to test the complete logic of the program for Example 11.5, the following set of data is necessary :

(a) Data for a male candidate.

(b) Data for a female candidate who has scored less than 45%.

(c) Data for a female candidate who has exactly scored 45%.

(d) Data for a female candidate who has scored more than 45% but less than 60%.

(e) Data for a female candidate who has exactly scored 60%.



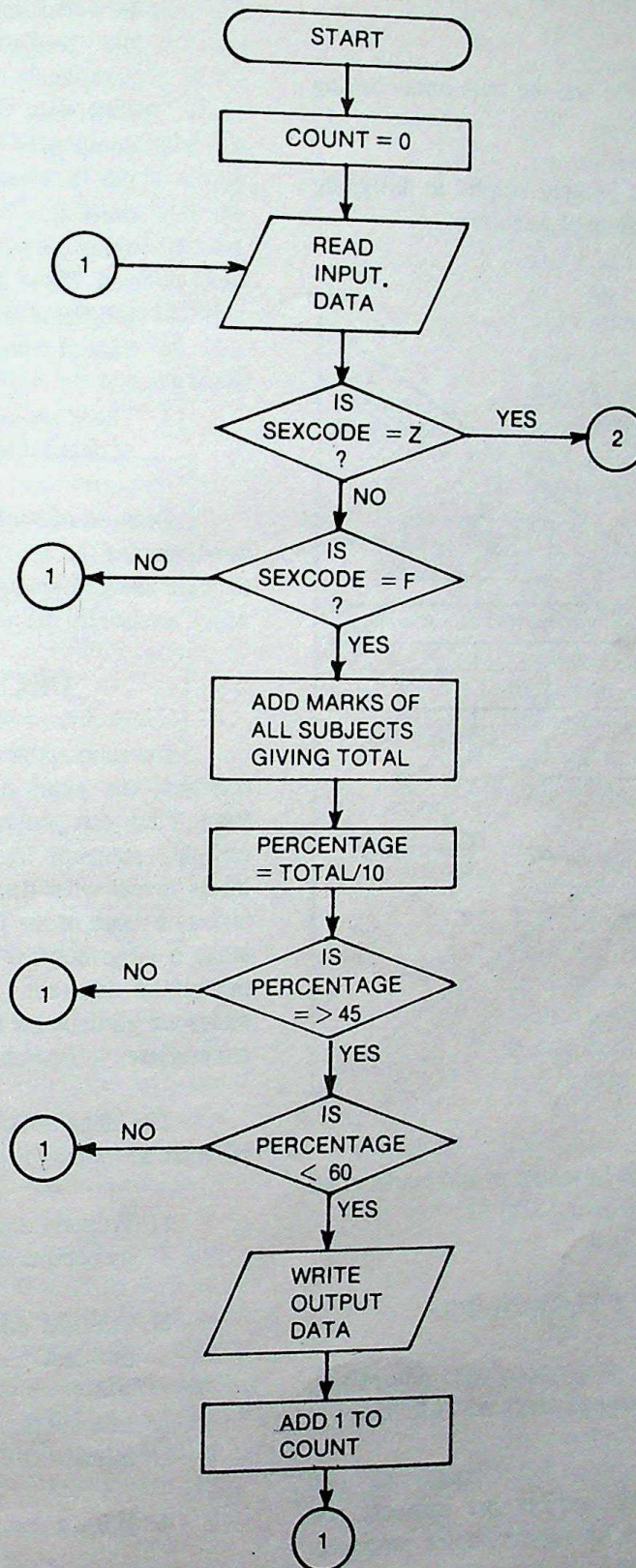


Figure 11.9. Flowchart of Figure 11.8 redrawn to illustrate the use of connectors  
 CC-0 Agamnigam Digital Preservation Foundation, Chandigarh. Funding by IKS-MoE-2025-Grant



- (f) Data for a female candidate who has scored more than 60%.
- (g) And obviously in the end the trailer data having sentinel value.

A flowchart proves to be very helpful in designing the test data for systematic testing of programs.

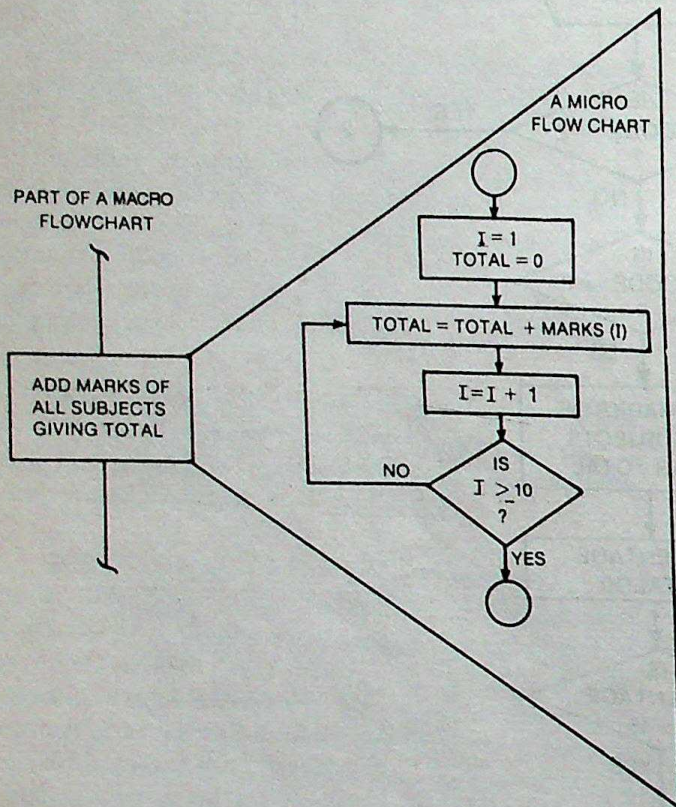


Figure 11.10. Detailed flowchart of add marks of all subjects giving TOTAL

### LIMITATIONS OF FLOWCHARTS

In spite of their many obvious advantages, flowcharts have several limitations which are as follows :

1. Flowcharts are very time consuming and laborious to draw with proper symbols and spacing, especially for large complex programs. In this chapter, you have seen examples of small program flowcharts developed for relatively small programs. You can very well imagine how difficult it would be to develop a detailed program flowchart for a program containing over 50,000 statements.

2. Owing to the symbol-string nature of flowcharting, any changes or modifications in the program logic will usually require a completely new flowchart. Redrawing a flowchart is again so tedious that many companies either do not redo them or produce the flowchart by using a computer program to draw it. There are several computer programs available that will read the program's instructions and draw a flowchart of its logic, but these programs are fairly expensive to acquire and use a lot of computer time.
3. There are no standards determining the amount of detail that should be included in a flowchart.

Because of such limitations, many organizations are now reducing the amount of flowcharting used. In its place, they are using alternative tools for program analysis, two of which are briefly discussed below.

### DECISION TABLES

Decision tables are used to define clearly and concisely the word statement of a problem in a tabular form. They can prove to be a powerful tool for defining complex program logic. As the name implies, decision tables reveal what decisions or actions the computer is to take as a result of the input data. When the computer has to make a large number of decisions or if there are a large number of different branches within a program, decision tables are particularly useful. In these cases, decision tables are preferred to flowcharts.

The steps to be followed for constructing a decision table are as follows :

1. Properly define the problem that has to be solved by computer.
2. List out all the conditions to be tested in the problem.
3. List out the corresponding actions that should be taken with each combination of conditions.
4. Form a decision table using the two lists.

Most decision tables have six parts. The basic format of a decision table is shown in Figure 11.11. The first part of the decision table contains the name and number of the table. For some larger applications, two or more decision tables may be used in the analysis part of program development. The second part of the table, known



as condition stub, describes the conditions that could exist in the program logic. Thus, the contents of condition stub correspond to the conditions contained in the decision symbols of a flowchart. Action stub, the third part of the decision table, contains the action statements. These statements correspond to the statements located in nondecision symbols of a flowchart. While the condition statements reveal the possible states of the input data, the action statements describe the possible actions of the computer system. The right hand side of the decision table contains the rule numbers (part 4), the actual conditions (part 5), and the actions taken by the computer (part 6). The condition entries correspond to the paths leading out from decision symbols.

| TABLE HEADING  | DECISION RULES    |
|----------------|-------------------|
| CONDITION STUB | CONDITION ENTRIES |
| ACTION STUB    | ACTION ENTRIES    |

Figure 11.11. Format of a decision table.

A decision table for the problem of Example 11.5 that was charted in Figure 11.8 is shown in Figure 11.12 In this table, each rule number is a given condition followed by a specific action to be taken by the computer. The six rule numbers, the six actual conditions, and the associated actions taken by the computer system are discussed below.

**Rule 1 :** The student is a female, and the percentage marks obtained is 45% or more, and the percentage marks obtained is less than 60%. The computer should write the output data, add 1 to COUNT, and then read the next student's record. It is a case of a female student who has passed in second division.

**Rule 2 :** The student is female, and the percentage marks obtained is 45% or more, and the percentage marks obtained is not less than 60%. The computer should directly read the next student's record without performing

any other operation. It is the case of a female student who has passed in first division (60% or more marks).

**Rule 3 :**

The student is a female and the percentage marks obtained is not 45% or more. A dash (-) in this column against the last condition means that this condition is irrelevant in arriving at the action. In fact if PERCENTAGE is not equal to or greater than 45 then it has to be less than 60. PERCENTAGE greater than or equal to 60 is logically impossible. Such conditions in the decision table which are irrelevant in arriving at an action are known as don't care conditions and are denoted by a dash. For this rule, the computer should directly read the next student's record without performing any other operation. It is the case of a female student who has scored less than 45% and hence she is not a second divisioner.

**Rule 4 :**

The student is a female, and the other conditions are don't care conditions. The computer should calculate PERCENTAGE after adding marks of all subjects and then proceed to test further conditions. It is the case of a female student whose PERCENTAGE has yet to be calculated.

**Rule 5 :**

The student is a male, and the other conditions are don't care conditions. The computer should directly read the next student's record without performing any other operation. It is the case of a male student.

**Rule 6 :**

In all the previous five rules, it was ascertained that the current record is not a trailer record because all these rules have a value N (No) for the condition SEXCODE = Z. In this rule, the SEXCODE is equal to Z which indicates a trailer record and hence the computer should write the value of COUNT and then STOP. The other conditions in this rule are don't care conditions. It is the case of a trailer record.



In the decision table, "Y" means yes, "N" means no, "-" means don't care, and "X" means the computer should take this action.

| Examination problem of Example 11.5 | Decision rule number. |   |   |   |   |   |
|-------------------------------------|-----------------------|---|---|---|---|---|
|                                     | 1                     | 2 | 3 | 4 | 5 | 6 |
| <b>Condition Statements</b>         |                       |   |   |   |   |   |
| SEXCODE = Z                         | N                     | N | N | N | N | Y |
| SEXCODE = F                         | Y                     | Y | Y | Y | N | - |
| PERCENTAGE = > 45                   | Y                     | Y | N | - | - | - |
| PERCENTAGE < 60                     | Y                     | N | - | - | - | - |
| <b>Actions Taken</b>                |                       |   |   |   |   |   |
| Calculate PERCENTAGE                |                       |   |   | X |   |   |
| Write output data                   | X                     |   |   |   |   |   |
| Add 1 to COUNT                      | X                     |   |   |   |   |   |
| Read next student record            | X                     | X | X |   | X |   |
| Write COUNT                         |                       |   |   |   |   | X |
| STOP                                |                       |   |   |   |   | X |

Figure 11.12. Decision table for the examination problem of Example 11.5.

### ADVANTAGES AND LIMITATIONS OF DECISION TABLES

Decision tables are normally used in place of flowcharts because of the following reasons:

1. They are easier to draw and change than charts.
2. They provide more compact documentation. A small table can replace several pages of charts.

3. It is also easier to follow a particular path down one column than through several flowchart pages.

However, decision tables are not very popular and are not so widely used as charts because :

1. Flowcharts are better able to express the total sequence of events needed to solve a problem.
2. Flowcharts are more familiar to, and are preferred by, many programmers and beginners

## PSEUDOCODE

Pseudocode is another programming analysis tool that is used for planning program logic. "Pseudo" means imitation or false and "Code" refers to the instructions written in a programming language. Pseudocode, therefore, is an imitation of actual computer instructions. These pseudoinstructions are phrases written in ordinary natural language (e.g., English, French, German, etc.). Instead of using symbols to describe the logic steps of a program, as in flowcharting, pseudocode uses a structure that resembles computer instructions. Because it emphasises the design of the program, pseudocode is also called Program Design Language (PDL).

Pseudocode is made up of the following basic logic structures that have been proved to be sufficient for writing any computer program :

1. Sequence
2. Selection (IF. . THEN. . ELSE or IF. . THEN)

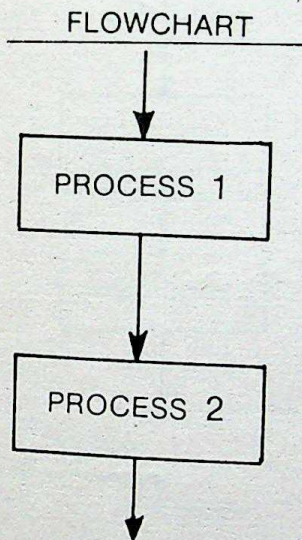
3. Iteration (DO...WHILE or REPEAT...UNTIL)

Sequence logic is used for performing instructions one after another in sequence. Thus, for sequence logic, pseudocode instructions are written in the order, in which they are to be performed. The logic flow of pseudocode is from the top to the bottom. Figure 11.13 shows an example of sequence logic structure.

Selection logic, also known as decision logic, is used for making decisions. It is used for selecting the proper path out of the two or more alternative paths in the program logic. Selection logic is depicted as either an IF...THEN or IF...THEN...ELSE structure. The flowcharts of Figure 11.14 and 11.15 illustrate the logic of these structures.



Their corresponding pseudocode is also given in these figures. The IF...THEN...ELSE construct says that if the condition is true, then do process 1, else (if the condition is not true) do process 2. Thus, in this case either process 1 or process 2 will be executed depending on whether the specified condition is true or false. However, if we do not want to choose between two processes and if we simply want to decide if a process is to be performed or not, then the IF...THEN structure is used. The IF...THEN structure says that if the condition is true, then do process 1 and if it is not true then skip over process 1. In both the structures, process 1 and process 2 can actually be one or more processes. They are not limited to a single process. ENDIF is used to indicate the end of the decision structures.



#### PSEUDOCODE

•

•

•

Process 1

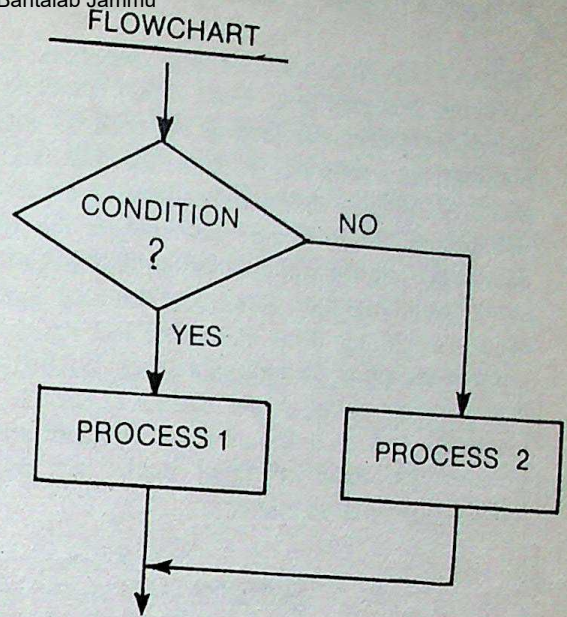
Process 2

•

•

•

Figure 11.13. Flowchart and pseudocode for sequence structure.



#### PSEUDOCODE

•

•

•

IF Condition

THEN process 1

ELSE

process 2

END IF

•

•

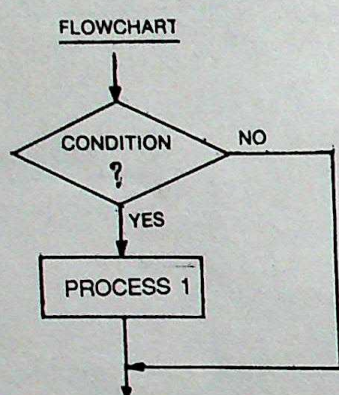
•

Figure 11.14. Flowchart and pseudocode for IF...THEN...ELSE selection structure.

Iteration logic is used to produce loops when one or more instructions may be executed several times depending on some condition. It uses two structures called the DO...WHILE and the REPEAT...UNTIL. They are illustrated by flowcharts in Figure 11.16 and Figure 11.17 respectively. Their corresponding pseudocodes are also given in these figures. Both DO...WHILE and REPEAT...UNTIL are used for looping. The differences are that in the DO...WHILE, the looping will continue as long as the condition is true. The looping stops when the condition is not true. On the other hand, in case of



REPEAT...UNTIL, the looping continues until the condition becomes true. That is, the execution of the statements within the loop is repeated as long as the condition is not true. In both the DO...WHILE and REPEAT...UNTIL, the loop must contain a statement that will change the condition that controls the loop. If it doesn't, the looping will continue without end which is the case of an infinite loop. Remember that no program should contain an infinite loop. Also note that the condition is tested at the top of the loop in the DO...WHILE and at the bottom of the loop in the REPEAT...UNTIL. ENDDO marks the end of a DO...WHILE structure and UNTIL followed by some condition marks the end of the REPEAT...UNTIL structure.



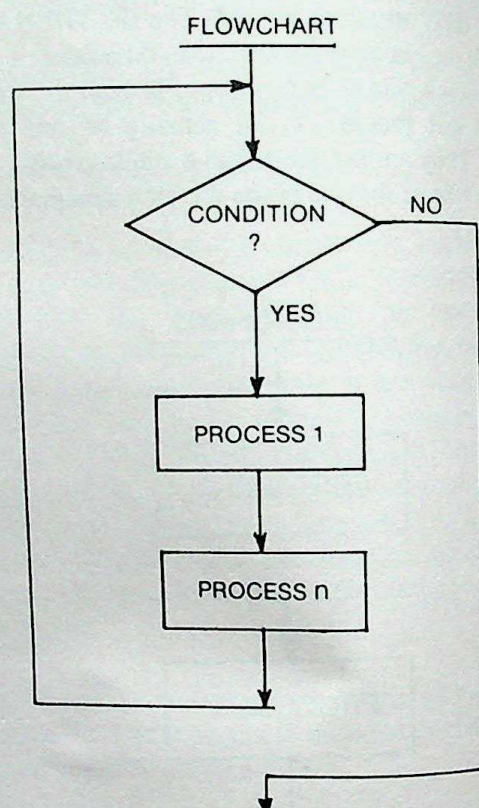
PSEUDOCODE

- 
- 
- 
- IF CONDITION
- THEN PROCESS 1
- END IF
- 
- 
- 

Figure 11.15. Flowchart and pseudocode for IF...THEN selection structure.

The pseudocode version of the logic of the problem of Example 11.5 that was charted in Figure 11.8 is shown in Figure 11.18. In the pseudocode example, the first line initializes the value of COUNT to zero and the second line reads the input data of the first student. The third line is the beginning of a loop using the DO...WHILE structure. It indicates that the loop will continue so long as the value of SEXCODE is not equal to Z - that is, as long as the trailer record is not found. In this example, a series of decisions followed by an instruction to read next student's record are included within the loop. The first statement within the

DO...WHILE loop asks, "Is the SEXCODE equal to F?" If the answer is yes, PERCENTAGE is calculated and again the third statement within the loop asks, "Is PERCENTAGE equal to or greater than 45?" If it is, the "Is PERCENTAGE less than 60?" This is a series of three IF...THEN decision structures. Each one ends with an ENDIF vertically aligned below the appropriate IF.

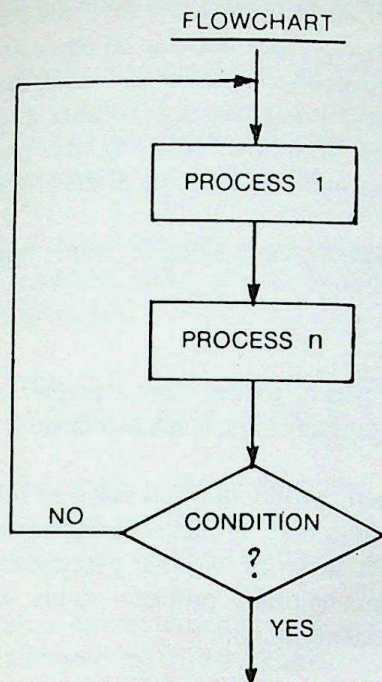


PSEUDOCODE

- 
- 
- 
- DO WHILE CONDITION
- PROCESS 1
- PROCESS n
- ENDDO
- 
- 
- 

Figure 11.16. Flowchart and pseudocode for DO...WHILE iteration structure.





PSEUDOCODE

•  
•  
•

REPEAT

PROCESS 1

PROCESS 2

UNTIL CONDITION

•  
•  
•

DO...WHILE loop to stop, because the condition (SEXCODE not equal to Z) is no longer true. When the DO...WHILE condition is no longer true, the next logical step will be the instruction following the ENDDO. At this stage, the value of COUNT will be printed (write COUNT) and finally the program execution will stop (STOP).

```

Set COUNT to zero
Read first student record
DO WHILE SEXCODE is not equal to Z
  IF SEXCODE = F THEN
    calculate PERCENTAGE
    IF PERCENTAGE => 45 THEN
      IF PERCENTAGE < 60 THEN
        Write output data
        Add 1 to COUNT
      ENDIF
    ENDIF
  ENDIF
  Read next student record
ENDDO
Write COUNT
STOP
    
```

Figure 11.18. Pseudocode for the examination problem of Example 11.5.

Figure 11.17. Flowchart and pseudocode for REPEAT...UNTIL iteration structure.

The two instructions - write output data, and add 1 to COUNT are performed only if all three conditions (that SEXCODE being F, PERCENTAGE being equal to or more than 45, and PERCENTAGE being less than 60) are found to be true (answered yes). If any of the three conditions is not true, the logic path goes to the statement that reads next student's record. After the last student's record is processed, the next record for which the value of SEXCODE is Z is encountered. This will cause the

One important feature of pseudocode as a programming tool is the use of indentation. Each statement within the DO...WHILE loop is indented, to show that it is part of the loop. Similarly, the statements within each IF...THEN structure is indented properly to clearly specify the statements which are part of each structure. The use of indentation in pseudocode is the same technique used with the various programming languages. Its sole purpose is to clarify the logical structure of the program. With this technique, we can tell at a glance which statements make up each of the logic structure of the total program logic. To



fully appreciate this factor, the reader should compare the equivalent non-indented pseudocode of Figure 11.19 to that of Figure 11.18. The difference in clarity would be far greater if this were a longer pseudocode covering, for instance, one or more pages.

### ADVANTAGES AND LIMITATIONS OF PSEUDOCODE

Pseudocode has three main advantages:

1. Converting a pseudocode to a programming language is much more easier as compared to converting a flowchart or a decision table.
2. As compared to a flowchart, it is easier to modify the pseudocode of a program logic when program modifications are necessary.

SET COUNT to Zero

Read first student record

DO WHILE SEXCODE is not equal to Z.

IF SEXCODE = F

THEN Calculate PERCENTAGE

IF PERCENTAGE = > 45

THEN IF PERCENTAGE < 60

THEN Write output data

Add 1 to COUNT

ENDIF

ENDIF

ENDIF

Read next student record

ENDDO

Write COUNT

STOP

3. Writing of pseudocode involves much less time and effort than drawing an equivalent flowchart. Pseudocode is easier to write than an actual programming language because it has only a few rules to follow, allowing the programmer to concentrate on the logic of the program.

However, pseudocode suffers from the following limitations :

1. In case of pseudocode, a graphic representation of program logic is not available.
2. There are no standard rules to follow in using pseudocode. Different programmers use their own style of writing pseudocode and hence a communication problem occurs due to lack of standardization.
3. For a beginner, it is more difficult to follow the logic of or write pseudocode, as compared to flowcharting.

### QUESTIONS

1. Why is it advisable to plan the logic of a program before writing it ?
2. What is an algorithm? What are the characteristics necessary for a sequence of instructions to qualify as an algorithm ?
3. What is a flowchart ?
4. How does a flowchart help a programmer in program development ?
5. Can a flowchart be drawn for a task if the person drawing the flowchart cannot perform the task manually ? Discuss.
6. What are the various symbols used in flowcharting ? Give their pictorial representation.
7. Describe the function of the various flowchart symbols.
8. Why are there standards for the symbols used in drawing flowcharts ?
9. What is a record ? A trailer record ?

Figure 11.19. Non-indented version of pseudocode of Figure 11.18.



10. What is a sentinel value ? Discuss its use.
11. What is a process loop ? An infinite loop ?
12. Why is it necessary to avoid infinite loops in program design ?
13. "A loop consists of a body, a test for exit condition, and a return provision." Discuss this statement.
14. What is a generalized algorithm ? Why should programs be general in nature ?
15. Discuss the difference between loop control by counting and loop control by the use of sentinel value. Which is preferable and why ?
16. How can a counter be used to keep track of the number of times a loop has been executed ?
17. Is it possible to have more than one flowchart for a given problem ? Give reasons for your answer.
18. What is the difference between a macroflowchart and a microflowchart? Illustrate with an example.
19. What are the various guidelines to be followed while drawing a flowchart ?
20. Discuss the various advantages and limitations of flowcharting.
21. Why is proper documentation of a program necessary ?
22. What are program bugs ? What is debugging ?
23. What is meant by testing a program ? How is it done ?
24. What are decision tables? When are they preferred to flowcharts ?
25. What are the various steps to be followed for constructing a decision table ?
26. Draw the basic format of a decision table and discuss the role of each part.
27. What are don't care conditions in decision tables ?
28. List out the various advantages and limitations of decision tables.
29. What is a pseudocode ? Why is it so called ? Give another name for pseudocode.
30. What are the three basic logic structures used in writing pseudocode ? Discuss the use of each.
31. Draw flowcharts for the two different structures used for selection logic.
32. What is the difference between the IF...THEN and the IF...THEN...ELSE structures ?
33. Draw flowcharts for the two different structures used for iteration logic.
34. Both DO...WHILE and REPEAT...UNTIL are used for looping. Discuss the difference between the two structures.
35. What is the purpose of the ENDIF and ENDDO?
36. What is indentation ? Why is it used in writing pseudocodes ?
37. Discuss the advantages and limitations of pseudocode.
38. Three numbers, denoted by the variables A, B, and C are supplied as input data. Draw a flowchart for the logic to pick and print the largest of the three numbers.
39. Draw a flowchart of the logical steps needed to produce a printed listing of all students over the age of 20 in a class. The input records contain the name and age of the students. Assume a sentinel value of 99 for the age field of the trailer record.
40. Draw a flowchart of the logical steps needed to print the name and age of the oldest and the youngest student in a class. The input records contain the name and age of the students. Assume a sentinel value of 99 for the age field of the trailer record.
41. The first 20 records in a data set are to be read and printed. Draw a flowchart for the algorithm to do this job. Make sure that the processing stops after the twentieth record.



42. Input data regarding the information of employees of a company has been supplied. The first field of each input record contains the employee number (EMPNO). Assume that the input data of all the employees is terminated by a trailer record having a trailer value of 99999 for EMPNO. Draw a flowchart for the logic to count and print the total number of input records, that is, the total number of employees.
43. For the employees problem of Question 42, we want to count and print the number of only male employees in the age range of 25 to 30. Assume that the input records contain SEXCODE and AGE fields to provide this information. Draw a flowchart for the algorithm to perform this job.
44. Suppose that a population survey has been carried out in a given city, and that the information received from the survey has been transcribed onto punched cards. Since the cards have 80 columns each, one card contains the name, address, sex, age, profession, etc., of one employee. That is, each card contains one record pertaining to one employee. Our problem is to print the details of all the adults (aged 18 years or more) in the city under survey. Finally, we also want to print the total number of adults. Assume a suitable sentinel value for any field in the trailer record and draw a flowchart for the algorithm to do this task.
45. A set of examination papers which have been graded with scores from 0 to 100 is to be searched to find how many of them are above 90. The total has to be printed. Prepare a flowchart to illustrate this job. Assume a suitable sentinel value for the trailer record.
46. Each paper in a set of examination papers includes a grade of A, B, C, D, or E. A count is to be made of how many papers have grade of A and how many have grade of E. The total count of both types have to be printed at the end. Prepare a flowchart to perform this function. Assume a suitable sentinel value for the trailer record.
47. A shopkeeper wants to have a general program for his personal computer that will prepare bills for each customer as and when he sells goods to them. His idea is that as soon as the customer purchases some goods from his shop, he will supply the description, unit price, and the quantity purchased for each item, as input to the computer. He wants that with this information, the computer should print each item along with its unit price, quantity purchased and the total price. Finally, the computer should also print the total cost of all the items purchased by the customer. Assuming that a sentinel value of zero is used for the quantity purchased field in the trailer record, draw a flowchart for the logic to do this job.
48. An employer plans to pay a bonus to each employee. Those earning Rs.2000 or above are to be paid 10 percent of their salary; those earning less than Rs.2000 are to be paid Rs.200. The input records contain the employee number, name, and salary of the employees. The output to be printed should contain the employee number, name, and the amount of bonus to be paid for each employee. Draw a flowchart for the algorithm to do this job. Assume a suitable sentinel value for any of the fields of the trailer record.
49. Each employee pay record includes the hours worked and the pay rate. The gross pay is to be determined as hours worked times pay rate and is to be printed for each employee. For all hours worked in excess of 40, the overtime rate, which is 1.5 times the regular rate, is to be paid. Draw a flowchart to illustrate the problem logic. Assume a suitable sentinel value for any of the input fields of the trailer record.
50. A data file contains a set of examination scores and is followed by a trailer record with a value of -1. Draw a flowchart for the logic to calculate and print the average of the scores.
51. The data file of Question 50 is expanded to include several sets of data, each requiring calculation of its average. Each data set is followed by a trailer record with a value of -1; however, the last data set is followed by a trailer record with a value of -2. Draw a flowchart for the logic to perform this task.
52. Suppose five numbers denoted by the variables A, B, C, D, and E are supplied as input. Draw a flowchart for the logic to print these numbers in descending order of magnitude.
53. Draw a flowchart for the logic to find whether a given triangle ABC is isosceles.



Assume that the angles of the triangle are supplied as input. Print the answer as yes or no.

54. Draw a flowchart for the logic to find out whether a given triangle ABC is a right angled triangle. Assume that the sides of the triangle are supplied as input data. Print the answer as yes or no.

55. Draw a flowchart for the logic to find out whether a given quadrilateral ABCD is a rectangle. Assume that all the four angles and four sides of the quadrilateral are supplied as input data. Print the answer as yes or no.

56. Draw a flowchart to illustrate the logic to convert a number from base 10 to a new base using the division-remainder technique.

57. Draw a flowchart to illustrate the logic to convert a number to decimal from another base.

58. Obtain a decision table to solve the problem described in Question 38.

59. Obtain a decision table to solve the problem described in Question 43.

60. Obtain a decision table to solve the problem described in Question 46.

61. Obtain a decision table to solve the problem described in Question 53.

62. Obtain a decision table to solve the problem described in Question 54.

63. Write the pseudocode to solve the problem described in Question 39.

64. Write the pseudocode to solve the problem described in Question 40.

65. Write the pseudocode to solve the problem described in Question 43.

66. Write the pseudocode to solve the problem described in Question 46.

67. Write the pseudocode to solve the problem described in Question 47.

68. Write the pseudocode to solve the problem described in Question 48.

69. Write the pseudocode to solve the problem described in Question 49.

70. Write the pseudocode to solve the problem described in Question 53.







## CHAPTER 12

# 12. COMPUTER LANGUAGES

This chapter continues the development of computer programs that was begun in Chapter 11. After the programming analysis phase, discussed in the previous chapter, has been completed, the second step in the development of computer programs is to write the specific instructions, needed to process an application, into a language and form acceptable to a computer system. The process of writing such program instructions for an analysed problem is called *coding*. In this chapter, we will see how the logical steps of our program plan will be coded as program instructions. The goal of this chapter is to introduce some of the common computer languages used in writing computer programs.

## ANALOGY WITH NATURAL LANGUAGES

A language is a system of communication. With our natural language such as English, we communicate to one another our ideas and emotions. Similarly, a computer language is a means of communication used to communicate between people and the computer. With the help of a computer language, a programmer tells a computer what he wants it to do. All natural languages

(English, French, German, etc.) use a standard set of symbols for the purpose of communication. These symbols are understood by everyone using that language. We normally call this set of symbols the vocabulary of that particular language. For example, the words we use in English are the symbols of English language that make up its vocabulary. Each word has definite meaning which can be looked up in a dictionary. In a similar manner, all computer languages have a vocabulary of their own. Each symbol of the vocabulary has definite unambiguous meaning which can be looked up in the manual meant for that language. Hence, each symbol of a computer language is used to tell the computer to do a particular job. The main difference between a natural language and a computer language is that natural languages have a large vocabulary but most computer languages use a very limited or restricted vocabulary. This is mainly because a programming language by its very nature and purpose does not need to say too much. Each and every problem to be solved by a computer has to be broken down into discrete (simple and separate), logical steps which basically comprise of four fundamental operations - input and output operations, arithmetic operations, movement of information within the CPU, and logical or comparison operations.



Each natural language has a systematic method of using symbols of that language. In English, this method is given by the rules of grammar. These rules tell us which words to use and how to use them. Similarly, the symbols of a particular computer language must also be used as per set rules which are known as the syntax rules of the language. In case of a natural language, people can use poor or incorrect vocabulary and grammar and still make themselves understood. However, computers, being a machine, are receptive only to exact vocabulary used correctly as per syntax rules of the language being used. Thus, in case of a computer language, we must stick by the exact rules of the language if we want to be understood by the computer. As yet, no computer is capable of correcting and deducing meaning from incorrect instructions. Computer languages are smaller and simpler than natural languages but they have to be used with great precision.

Unless a programmer adheres exactly to the syntax rules of a programming language, even down to the correct punctuation marks, his commands will not be understood by the computer.

Programming languages have improved throughout the years, just as computer hardware has improved. They have progressed from machine-oriented languages that use strings of binary 1s and 0s to problem-oriented languages that use common mathematical and/or English terms. However, all computer languages can be classified in the following three broad categories :

- (a) Machine Language
- (b) Assembly Language
- (c) High-level Language.

We shall now examine the evolution and nature of each type of language.

## MACHINE LANGUAGE

Although computers can be programmed to understand many different computer languages, there is only one language understood by the computer without using a translation program. This language is called the *machine language* or the *machine code* of the computer. Machine code is the fundamental language of a computer and is normally written as strings of binary 1s and 0s. The circuitry of a computer is wired in such a way that it immediately recognizes the machine language and converts

it into the electrical signals needed to run the computer.

An instruction prepared in any machine language has a two-part format, as shown in Figure 12.1. The first part is the command or operation, and it tells the computer what function to perform. Every computer has an *operation code* or *opcode* for each of its functions. The second part of the instruction is the operand, and it tells the computer where to find or store the data or other instructions that are to be manipulated. Thus, each instruction tells the control unit of the CPU what to do and the length and location of the data fields that are involved in the operation. Typical operations involve reading, adding, subtracting, writing, and so on.

| OPCODE<br>(operation code) | OPERAND<br>(Address/Location) |
|----------------------------|-------------------------------|
|----------------------------|-------------------------------|

Figure 12.1. Instruction format.

We already know that all computers use binary digits (0s and 1s) for performing internal operations. Hence, most computers' machine language consists of strings of binary numbers and is the only one the CPU directly understands. When stored inside the computer, the symbols which make up the machine language program are made up of 1s and 0s. For example, a typical program instruction to print out a number on the printer might be

101100111111010011101100

The program to add two numbers in memory and print the result might look something like the following :

00100000000001100111001  
001100000000010000100001  
011000000000011100101110  
10100111111011100101110  
000000000000000000000000

This is obviously not a very easy language to learn, partly because it is difficult to read and understand and partly because it is written in a number system with which we are not familiar. But it will be surprising to note that some of



the first programmers, who worked with the first few computers, actually wrote their programs in binary form as above.

Since human programmers are more familiar with the decimal number system, most of them preferred to write the computer instructions in decimal, and leave the input device to convert these to binary. In fact, without too much effort, a computer can be wired so that instead of using long strings of 1s and 0s we can use the more familiar decimal numbers. With this change, the preceding program appears as follows :

```
10001471
14002041
30003456
50773456
00000000
```

The set of instruction codes, whether in binary or decimal, which can be directly understood by the CPU of a computer without the help of a translating program, is called a machine code or machine language. Thus, a machine language program need not necessarily be coded as strings of binary digits (1s and 0s). It can also be written using decimal digits if the circuitry of the computer being used permits this.

### Advantages and Limitations of Machine Language

Programs written in machine language can be executed very fast by the computer. This is mainly because machine instructions are directly understood by the CPU and no translation of the program is required. However, writing a program in machine language has several disadvantages which are discussed below.

1. *Machine dependent.* Because the internal design of every type of computer is different from every other type of computer and needs different electrical signals to operate, the machine language also is different from computer to computer. It is determined by the actual design or construction of the ALU, the control unit, and the size as well as the word length of the memory unit. Hence, suppose after becoming proficient in the machine code of a particular computer, a company decides to change to another computer, the programmer may be required to learn a new machine language and would have to rewrite all the existing programs.

2. *Difficult to program.* Although easily used by the computer, machine language is difficult to program. It is necessary for the programmer either to memorize the dozens of code numbers for the commands in the machine's instruction set or to constantly refer to a reference card. A programmer is also forced to keep track of the storage location of data and instructions. Moreover, a machine language programmer must be an expert who knows about the hardware structure of the computer.
3. *Error prone.* For writing programs in machine language, since a programmer has to remember the opcodes and he must also keep track of the storage location of data and instructions, it becomes very difficult for him to concentrate fully on the logic of the problem. This frequently results in program errors. Hence, it is easy to make errors while using machine code.
4. *Difficult to modify.* It is difficult to correct or modify machine language programs. Checking machine instructions to locate errors is about as tedious as writing them initially. Similarly, modifying a machine language program at a later date is so difficult that many programmers would prefer to code the new logic afresh instead of incorporating the necessary modifications in the old program.

In short, writing a program in machine language is so difficult and time consuming that it is rarely used today.

## ASSEMBLY LANGUAGE

One of the first steps in improving the program preparation process was to substitute letter symbols-mnemonics for the numeric operation codes of machine language. A *mnemonic* (or memory aid) is any kind of mental trick we use to help us remember. Mnemonics come in various shapes and sizes, all of them useful in their own way. For example, a computer may be designed to interpret the machine code of 1111 (binary) or 15 (decimal) as the operation 'subtract', but it is easier for a human being to remember it as SUB.

### Use of Symbols Instead of Numeric OpCodes

All computers have the power of handling letters as well as numbers. Hence, a computer can be taught to recognize certain combination of letters or numbers. It can be taught (by means of a program) to substitute the number



14 every time it sees the symbol ADD, substitute the number 15 every time it sees the symbol SUB, and so forth. In this way, the computer can be trained to translate a program written with symbols instead of numbers into the computer's own machine language. Then we can write program for the computer using symbols instead of numbers, and have the computer do its own translating. This makes it easier for the programmer, because he can use letters, symbols, and mnemonics instead of numbers for writing his programs. For example, the preceding program that was written in machine language for adding two numbers and printing out the result could be written in the following way :

```
CLA A
ADD B
STA C
TYP C
HLT
```

Which would mean "take A, add B, store the result in C, type C, and halt." The computer, by means of a translating program, would translate each line of this program into the corresponding machine language program.

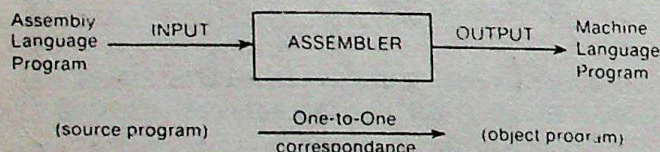


Figure 12.2. Illustrating the translation process of an assembler.

At this point we must learn a few more terms. The language which substitutes letters and symbols for the numbers in the machine language program is called an *assembly language* or *symbolic language*. A program written in symbolic language that uses symbols instead of numbers is called an *assembly code* or a *symbolic program*. The translator program that translates an assembly code into the computer's machine code is called an *assembler*. The assembler is a system program which is supplied by the computer manufacturer. It is written by system programmers with great care. It is so called because in addition to translating the assembly code into machine

code, it also, 'assembles' the machine code in the main memory of the computer and makes it ready for execution. (A symbolic program written by a programmer in assembly language is called a *source program*.) After the source program has been converted into machine language by an assembler, it is referred to as an *object program*. As shown in Figure 12.2, the input to an assembler is a source program written in assembly language and its output is an object program which is in machine language. Since the assembler translates each assembly language instruction into an equivalent machine language instruction, there is a one-to-one correspondence between the assembly instructions of source program and the machine instructions of object program.

By now, it must have been clear to the readers that when we write a program in symbolic language, we first run the assembler (program) to assemble the symbolic program into machine language, and then we run the machine language program to get our answer. You will notice that this means more time spent by the computer - it not only has to run the main program to get the answer, but it also must first translate the original symbolic program into machine language. But symbolic programming saves so much time and effort of the programmer that the extra time spent by the computer is worth it.

To see how symbolic programming works, let us first write a short machine language program and then see how we would write the same program in assembly language. For this, let us assume that the computer uses the following mnemonics for the operation codes mentioned against

| Mnemonic | Op Code | Meaning                              |
|----------|---------|--------------------------------------|
| HLT      | 00      | Halt, used at end of program to stop |
| CLA      | 10      | Clear & Add into A register.         |
| ADD      | 14      | Add to the contents of A register    |
| SUB      | 15      | Subtract from contents of A register |
| STA      | 30      | Store A register                     |

each. For simplicity, here we have considered only five operation codes that will be used in writing our program. Like this there can be more than hundred operation codes available with a particular computer.

The program we will write is quite simple : adding two numbers and storing the sum. The regular machine language program for this will be as follows :



| Memory Location | Contents |         | comments                                        |
|-----------------|----------|---------|-------------------------------------------------|
|                 | op Code  | Address |                                                 |
| 0000            | 10       | 1000    | Clear & add first number to A register          |
| 0001            | 14       | 1001    | Add second number to the contents of A register |
| 0002            | 30       | 1002    | Store the answer from A register                |
| 0003            | 00       |         | Halt                                            |
| .               |          |         |                                                 |
| .               |          |         |                                                 |
| .               |          |         |                                                 |
| 1000            |          |         | Reserved for first number                       |
| 1001            |          |         | Reserved for second number                      |
| 1002            |          |         | Reserved for the answer.                        |

| Memory Location | Contents |         |
|-----------------|----------|---------|
|                 | Mnemonic | Address |
| 0000            | CLA      | 1000    |
| 0001            | ADD      | 1001    |
| 0002            | STA      | 1002    |
| 0003            | HLT      |         |

### Use of Symbols Instead of Addresses

But there is no need to stop here; after all, it is not much more work to memorize a few numbers for op-codes, and using symbols for op-codes just makes more work for the computer. We might as well let the computer do a large share of the work.

One of the greatest problems in machine language programming comes from keeping track of addresses. Every time we write a large program, we need a pad of paper off to the side on which we keep a running list of what numbers are stored where. Each time we want a number, we must look up the address on this list. This takes time and can lead to mistakes if we are not careful. Why not let the computer do this part of work for us? We can include, as part of the assembly program, a section that does nothing but keeps a list of addresses for numbers.

In the preceding sample program to add two numbers, we might start the symbolic program by telling the computer something like the following :

"From now on address 1000 will be called FRST, address 1001 will be called SCND, and address 1002 will be called ANSR."

In response to this, the assembler sets up a table somewhere in the computer memory which looks something like the following:

FRST = 1000  
SCND = 1001  
ANSR = 1002

From now on, we just call numbers by their names. The assembler will look up the name in the table and provide the right address, thus saving us even more work. This means we can write our sample program as shown below:

It has been assumed here that the computer is capable of handling decimal numbers instead of only binary numbers. The two numbers to be added are stored in memory locations 1000 and 1001, and the answer obtained after adding the two numbers is to be stored in location 1002. The first instruction at location 0000 clears (makes zero) the A register (accumulator) and puts the contents of location 1000 (first number) in it. The second machine instruction at location 0001 adds the contents of location 1001 (second number) to the contents of A register (first number) and stores the sum in A register. The third instruction at location 0002 stores the answer from A register into memory location 1002. Finally, the fourth machine instruction at location 0003 stops the execution of the program.

Now we will see how to write the same program in assembly language. We can easily replace the op-code in each of the preceding instructions by the corresponding mnemonic and write the program as given below instead; and let the computer handle the rest. The assembler program would then translate CLA to 10, ADD to 14, STA to 30 and HLT to 00, thus producing the machine language program.



| Memory Location | Contents |         |
|-----------------|----------|---------|
|                 | Mnemonic | Address |
| 0000            | CLA      | FRST    |
| 0001            | ADD      | SCND    |
| 0002            | STA      | ANSR    |
| 0003            | HLT      |         |

The assembler then looks up the address for each of the numbers, translates the mnemonic op-code into the numerical op-code and comes out with a machine language program. For example, in the first instruction CLA FRST, the assembler translates the CLA into an op-code of 10 and looks up the address of FRST in the table. It finds the address as 1000, and so it puts the first machine language instruction as 10 1000. In the same way it translates the ADD SCND instruction into 14 1001, and so on, making the machine language program step by step.

But there is one more part of our work that we can hand over to the computer. We need not even tell the computer where to place each number and where to place each instruction, as we have been doing till now. Instead of saying something like "Place FRST in 1000, SCND in 1001, and ANSR in 1002," we need only tell the computer to start putting the numbers into memory starting with address 1000. In the same way we do not have to specify that the instruction CLA FRST goes into 0000, ADD SCND into 0001, and so on - we merely tell the computer to start the program at location 0000. The symbolic program would therefore go something like this :

START PROGRAM AT 0000 AND START DATA AT 1000

SET ASIDE AN ADDRESS FOR FRST

SET ASIDE AN ADDRESS FOR SCND

SET ASIDE AN ADDRESS FOR ANSR

CLA FRST

ADD SCND

STA ANSR

HLT

Then we start the assembler working on this symbolic program. We see that the first four steps are not really parts of the main program to add the two numbers; they are instructions to the assembler on what to do, and are called *pseudo-instructions*. The word pseudo, from the greek word meaning false, fake or pretended, describe them quite well : they are instructions that do not do anything in the main program, but only provide information to the assembler to tell it how we want the program assembled.

To see how the assembler changes this symbolic program into machine language, let us follow it step by step.

The first step tells the assembler that the program should start at address 0000 and each following instruction should be in the following address, and that the data (in this example this includes the FRST and SCND numbers, as well as the ANSR) should start at memory location 1000.

The next step tells the assembler to set aside an address for FRST. The assembler therefore picks the first free address in the group set aside for data, which is 1000, and will call it FRST from now on. Note that because we will refer to the address with the symbol FRST from now on, we really do not care what the address exactly is: whether it is 1000 or 1001 or 4253 makes no difference because the assembler will keep track of it.

The next step tells the assembler to set aside an address for SCND. The next free address after 1000 is 1001 (because 1000 is already taken by FRST), and so the assembler will supply the address 1001 every time we use the symbol SCND.

In the same way, the next step tells the assembler to set aside an address, this time 1002, for ANSR. This is the last pseudo-instruction, and the next instruction will start the program itself.

The next instruction is CLA FRST, which the computer translates into 10 1000 by translating CLA into 10 and looking up the address of FRST in its table which is 1000. Similarly, the assembler will translate the instruction ADD SCND into 14 1001 and the instruction STA ANSR into 30 1002. Finally it translates HLT into 00, thus providing the complete machine language program for the given assembly language program.



## Advantages of Assembly Language Over Machine Language

Assembly languages have the following advantages over machine languages :

1. *Easier to understand and use.* Assembly languages are easier to understand and use because mnemonics are used instead of numeric op-codes and suitable names are used for data. The use of mnemonics means that comments are usually not needed; the program itself is understandable. Symbolic programming also saves a lot of time and effort of the programmer because it is easier to write as compared to machine language programs.
2. *Easy to locate and correct errors.* While writing programs in assembly language, fewer errors are made, and those that are made are easier to find and correct because of the use of mnemonics and symbolic field names. Furthermore, assemblers are so designed that they automatically catch errors. If we use an invalid mnemonic or a name that has never been defined, the assembler will print out an error indication. For example, suppose one instruction in the symbolic program reads ADD AREA, and we forget to define what AREA is, the assembler will look through its table to find AREA and not finding it will indicate the error.
3. *Easier to modify.* Assembly language programs are easier for people to modify than machine-language programs. This is mainly because they are easier to understand and hence it is easier to locate, correct, and modify instructions as and when desired. Moreover, insertion or removal of certain instructions from the program does not require change in the address part of the instructions following that part of the program. This is required in case of machine language.
4. *No worry about addresses.* One of the greatest advantage of assembly language is that it eliminates worry about address for instructions and data. This is more important than it seems at first glance. Suppose we have written a long machine language program involving many steps and many references to itself within the program, such as looping, and address modifications, and so on. At the very end we may suddenly discover that we have left out an instruction in the middle.

If we insert that instruction, we will have to remember all the following instructions, and go through the entire program to check any references to other steps. This is a tedious job. But if we write the same program in symbolic language, we merely add the extra instruction, and the assembler will take care of the step numbering automatically.

5. *Easily relocatable.* Suppose that an assembly language program starts at address 1000 and we suddenly find that we have another program to be used with this program and this program also starts at location 1000. Obviously, one of the two programs will have to be rewritten to be moved somewhere else. In machine language, this can be a complicated job. But in case of assembly language, we merely have to change the first statement; for example instead of :

START PROGRAM AT 1000 AND START DATA AT 2000

we merely change this first statement to

START PROGRAM AT 3000 AND START DATA AT 4000

and run the symbolic program once more through the assembler. The equivalent machine language program will this time start at memory location 3000 instead of 1000, and there will be no conflict with the other program. In other words, using symbolic language we can easily move programs from one section of the memory to another; we say that assembly language programs are easily relocatable because their location is easily changed merely by changing the first instruction. This is not easily done with machine language programming.

6. *Efficiency of machine language.* In addition to the above mentioned advantages, an assembly language program also enjoys the efficiency of its corresponding machine code because there is one to one correspondance between the instructions of an assembly language program and its corresponding machine language program. Except for pseudo-instructions, which are simply instructions to the assembler, every other instruction of an assembly language program is translated into one machine language instruction. For every machine language instruction, there is a corresponding symbolic instruction.



instruction and for every symbolic instruction (except the pseudo-instructions) there is a corresponding machine instruction. In other words, the symbolic program will be just as long as the resulting machine language program. So leaving out the translation time required by the assembler, the actual execution time of an assembly language program and its equivalent machine language program (written independently) will be the same. The reason we are stressing this important fact is that there are languages (called macro-languages) in which a single instruction may get translated into an entire series of machine language instructions. Assembly language, in its basic form, is not one of these - there is one-to-one relationship between symbolic and machine languages.

### Limitations of Assembly Language

The following disadvantages of machine language are not solved by using assembly language :

1. *Machine dependent.* Because each instruction in the symbolic language is translated into exactly one machine language instruction, assembly languages are machine dependent. That is, they are designed for the specific make and model of the processor being used. A decision to change to another computer still usually requires learning a new language and the conversion of all existing programs - a very expensive undertaking.
2. *Knowledge of hardware required.* Since assembly languages are machine dependent, so the programmer must be aware of a particular machine's characteristics and requirements as the program is written. An assembly language programmer must know how his machine works and should have a good knowledge of the logical structure of his computer in order to write a good assembly language program.
3. *Machine level coding.* In case of an assembly language, instructions are still written at the machine-code level - that is, one assembler instruction is substituted for one machine-code instruction.

Machine and assembly languages being machine dependent, are referred to as *low-level languages*.

### Assembly Languages With Macro Instructions

In general, assembly languages are termed one-for-one in nature, that is, each assembly language instruction will result in one machine language instruction. However, quite often a certain set of machine language or assembly language instructions have to be used over and over. For example, three instructions, one after the other, might be needed to print out a number on a particular computer. These three instructions, always in the same order, might be used over and over in the same program. Instead of forcing the programmer to write out the set of three instructions every time he wants to print a number, we might as well design the assembler (program) in such a way so as to take care of these instructions. Every time the programmer gave the PRINT instruction, for example, the assembler would translate it into three machine language instructions instead of one, thus supplying the complete set of instructions required for printing.

Any instruction, such as PRINT, which gets translated into several machine language instructions, is called a *macro instruction*. There might be many such macro instructions permitted by a particular assembler. Thus, to speed up the coding process, assemblers were developed that could produce a variable amount of machine language instructions for each macro instruction of the assembly language program.

The use of macro instructions adds much work to the computer because the translation process becomes more than just changing each word into a number. The assembler must be able to supply the missing steps as well, but it means a tremendous saving of work for the programmer. The programmer gets relieved of the task of writing an instruction for every machine operation performed. It reduces the length of the programs he writes, cuts down on his errors, and simplifies his programming.

The macro instruction capability was developed very early in the evolution of computer languages. In fact, it is this concept of multiple machine instructions from one macro instruction around which today's machine-independent higher level languages are designed.

### HIGH-LEVEL LANGUAGE

We have already seen that writing of programs in machine language or assembly language requires a deep knowledge of the internal structure of the computer. While writing programs in any of these languages, a programmer has to remember all the 'operation codes' (numeric or mnemonic) of the computer and know in detail what each code does and how it affects the various registers of the



computer. However, we have also seen that in order to write a good computer program, the programmer should mainly concentrate on the logic of the problem rather than be concerned with the details of the internal structure of the computer. In order to facilitate the programmers to use computers without the need to know in detail the internal structure of the computer, high-level languages were developed.)

High-level languages, instead of being machine based, are oriented more towards the problem to be solved. These languages enable the programmer to write instructions using English words and familiar mathematical symbols. So it becomes easier for him to concentrate on the logic of his problem rather than getting involved in programming details. Obviously, the two-part format shown in Figure 12.1, that was required for writing instructions in machine language or assembly language, is not necessary for writing instructions in a high-level language. For example, let us consider the same problem of adding two numbers (FRST & SCND) and storing the sum in ANSR. We have already seen that three low-level (machine/assembly) instructions are required for performing this job. However, if we use a high-level language, say FORTRAN for instance, to instruct the computer to do this job, only one instruction need be written :

ANSR = FRST + SCND

This instruction is obviously very easy to understand and write because it resembles the familiar algebraic notation for adding two numbers :  $a = b + c$ .

High-level languages are basically symbolic languages that use English words and/or mathematical symbols rather than mnemonic codes. In other words, a high-level language is a symbolic language with nothing but macro-instructions. Every instruction which the programmer writes in a high-level language is translated into many machine language instructions. This is one-to-many translation and not one-to-one as in the case of assembly language. It is due to this reason that high-level languages are so called.

High-level languages are also known as *problem-oriented languages* because the macro instructions are especially picked to be useful for solving particular types of problems. Each such language is then best to solve a particular class of problems and may be completely useless for solving other types of problems. For example, if a high-level language is capable of handling business-type applications that consist of high input volume, relatively

little processing, and a high output volume, then the language is a *business-oriented language*. On the other hand, languages excellent at performing sophisticated computations but not adept at handling large data files are *mathematically-oriented languages*. Thus, a problem-oriented language is designed in such a way that its instructions may be written more like the language of the problem. For example, a scientist using a science-oriented language can use scientific formulas, and a business man with a business-oriented language can use business terms. Hence, high-level languages are generally easier to learn and write.

## COMPILERS

Since a computer hardware is capable of understanding only machine level instructions, so it is necessary to convert the instructions of a program written in high-level language to machine instructions before the program can be executed by the computer. We have seen that assembly languages use an assembler to perform this conversion process. In case of a high-level language, this job is carried out by a *compiler*. Thus, a compiler is a translating program (much more sophisticated than an assembler) that translates the instructions of a high-level language into machine language. A compiler is so called because it compiles a set of machine language instructions for every program instruction of a high-level language. A program written by a programmer in a high-level language is called a *source program*. After this source program has been converted into machine language by a compiler, it is referred to as an *object program*. As shown in Figure 12.3 the input to a compiler (program) is a source program written in a high-level language and its output is an object program which consists of machine language instructions. Note that the source program and the object program are the same program, but at different stages of development.

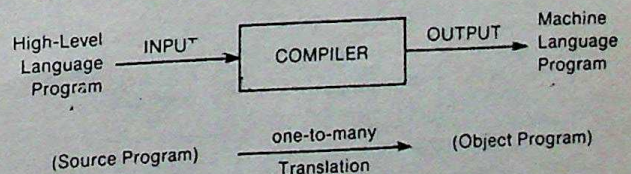


Figure 12.3. Illustrating the translation process of a compiler.

A compiler can translate only those source programs which have been written in the language for which the



computer is meant. For example, a FORTRAN compiler is only capable of translating source programs which have been written in FORTRAN and, therefore, each machine requires a separate compiler for each high-level language

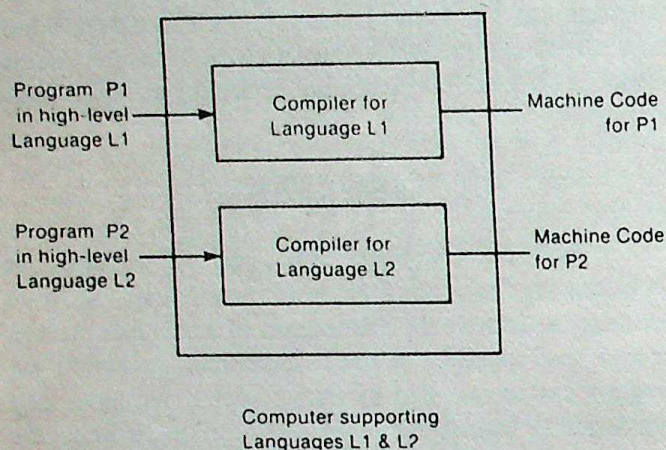


Figure 12.4. Illustrating the requirement of separate compilers for each high-level language supported by a computer.

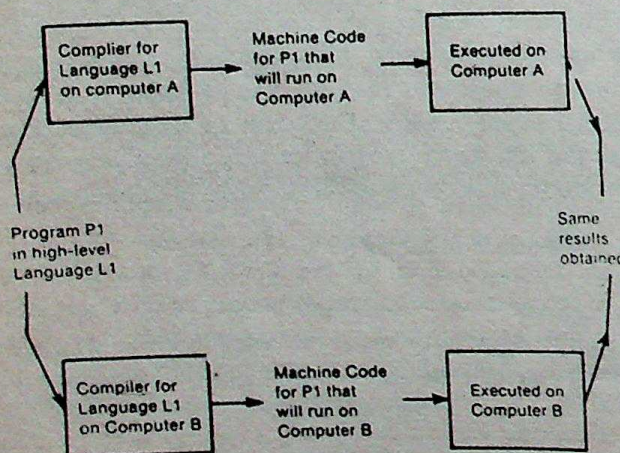


Figure 12.5. Illustrating the machine independence characteristic of a high-level language. Separate compilers are required for the same language on different computers.

that it supports. This is illustrated in Figure 12.4. Moreover, since an object program for one machine will not be the same as the object program for another machine, it is necessary that each machine must have its own 'personal' compiler for a particular language, say L1. Figure 12.5 illustrates how machine-independence is achieved by using different compilers to translate the same high-level language program to machine languages of different computers.

Compilers are large programs which reside permanently on secondary storage. When the translation of a program is to be done, they are copied into the main memory of the computer. The compiler, being a program, is executed in the CPU. While translating a given program, the compiler analyses each statement in the source program and generates a sequence of machine instructions which, when executed, will precisely carry out the computation specified in the statement. As the compiler analyses each statement, it uncovers certain types of errors. These are referred to as *diagnostic errors*. The compiler can diagnose the following kinds of errors in a source program :

- Illegal characters
- Illegal combination of characters
- Improper sequencing of instructions in a program.

A source program containing an error diagnosed by the compiler will not be compiled into an object program. The compiler will print out a suitable message indicating this, along with a list of coded error messages which indicate the type of errors committed. The error diagnostics is an invaluable aid to the programmer.

A compiler, however, cannot diagnose *logical errors*. It can only diagnose grammatical (syntax) errors in the program. It cannot know ones intentions. For example, if one has wrongly punched -25 as the age of a person, when he actually intended +25, the compiler cannot diagnose this. Programs containing such errors will be successfully compiled and the object code will be obtained without any error message. However, such programs, when executed, will not produce the right answers. So logical errors are detected only after the program is executed and the result produced does not tally with the desired result. Hence, it is essential to be precise in writing a program and pay careful attention to the smallest detail.



## INTERPRETERS

An *interpreter* is another type of translator used for translating high-level languages into machine code. It takes one statement of a high-level language and translates it into a machine instruction which is immediately executed. Translation and execution alternate for each statement encountered in the high-level language program. In other words, an interpreter translates one instruction, and the control unit executes the resulting machine code, the next instruction is translated, and the control unit executes the machine code instruction, and so on. This differs from a compiler which merely translates the entire source program into an object program and is not involved in its execution. In case of a compiler, the whole source program is translated into an equivalent machine language program. The object code, thus obtained, is permanently saved for future use and is used every time the program is to be executed. So repeated compilation is not necessary for repeated execution of a program. However, in case of an interpreter, no object code is saved for future use because the translation and the execution processes alternate. The next time an instruction is used, it must once again be interpreted and translated into machine language. For example, during the repetitive processing of the instructions within a loop, each instruction in the loop will have to be reinterpreted every time the loop is executed.

Interpreters are often employed with microcomputers (small computers). The advantage of an interpreter over a compiler is fast response to changes in the source program. The interpreter eliminates the need for a separate compiling run after each program change to add features or correct errors. Moreover, a compiler is a complex program compared to an interpreter. Interpreters are easy to write and they do not require large memory space in the computer. The interpreter, however, is a time consuming translation method because each statement must be translated every time it is executed from the source program. Thus, a compiled machine language program runs much faster than an interpreted program.

Assemblers, compilers, and interpreters are systems software that translate a source program written by the user to an object program which is meaningful to the hardware of the computer. These translators are also referred to as *language processors* since they are used for processing a particular language.

Based on the above discussions, we conclude that, in general, a programming language should possess the following characteristics to be considered high-level:

1. The language should be relatively independent of a given computer system. That is, instead of being machine based, it should be oriented more towards the problem to be solved.
2. Each statement of the language should be a macro instruction that gets translated into many machine language instructions.
3. The language should enable the programmers to write instructions using familiar words and mathematical symbols. It should be natural and should use abbreviations and words used in everyday communication.
4. The language should be independent of machine language instructions and other pieces of system software except for the compiler or the interpreter.
5. The language should not be experimental in nature and should exist on more than one computer system.

### Advantages of High-Level Languages

High level languages enjoy the following advantages over assembly and machine languages :

1. *Machine independence.* High-level languages are machine independent. This is a very valuable advantage because it means that a company changing computers - even to one from a different manufacturer - will not be required to rewrite all the programs that it is currently using. Even for programs written in the high-level languages, some modifications are almost always required, but these modifications are relatively minor and can be easily done without much effort. In other words, a program written in a high-level language can be run on many different types of computers with very little or practically no effort.
2. *Easy to learn and use.* These languages are very similar to the languages normally used by us in our day-to-day life. Hence they are easy to learn and use. The programmer need not learn anything about the computer he is going to use. He need not worry about how to store his numbers in the computer, where to store them, what to do with them, etc. That is the programmer need not know the machine instructions, the data format, and so on.



However, such a knowledge is desirable since it allows the programmer to utilize the system more efficiently.

3. *Fewer errors.* In case of high-level languages, since the programmer need not write all the small steps carried out by the computer, he is much less likely to make an error. The computer takes care of all the little details, and will not introduce any error of its own unless something breaks down. Furthermore, compilers are so designed that they automatically catch and point out the errors made by the programmer. Hence, diagnostic errors, if any, can be easily located and corrected by the programmer.
4. *Lower program preparation cost.* Writing programs in high-level languages requires less time and effort which ultimately leads to lower program preparation cost. Generally, the cost of all phases of program preparation (coding, debugging, testing, etc.) is lower with a high-level language than with an assembly language or with a machine language.
5. *Better documentation.* A high-level language is designed in such a way that its instructions may be written more like the language of the problem. Thus the statements of a program written in a high-level language can be easily understood by a person familiar with the problem. For the documentation of such programs, very few or practically no separate comment statements are required.
6. *Easier to maintain.* Programs written in high-level languages are easier to maintain than assembly language or machine language programs. This is mainly because they are easier to understand and hence it is easier to locate, correct, and modify instructions as and when desired. Insertion or removal of certain instructions from a program is also possible without any complication. Thus, major changes can be incorporated with very little effort.

### Limitations of High-level Languages

Two disadvantages of high-level languages are :

1. *Lower efficiency.* Generally, a program written in assembly language or machine language is more efficient than one written in high-level language. That is, the programs written in high-

level languages take more time to run and require more main storage.

2. *Lack of flexibility.* Because the automatic features of high-level languages always occur and are not under the control of the programmer, they are less flexible than assembly languages. An assembly language provides programmers access to all the special features of the machine they are using. Certain types of operations which are easily programmed using the machine's assembly language, are impractical to attempt using a high-level language. This lack of flexibility means that some tasks cannot be done in a high-level language, or can be done only with great difficulty.

In most cases, the advantages of high-level languages far outweigh the disadvantages. Most computer installations use a high-level language for most programs and use an assembly language for doing special tasks that cannot be easily done otherwise.

### SOME-HIGH-LEVEL LANGUAGES

The credit for the development of the first high-level language is usually given to Dr. Grace Hopper who described the idea of a compiler and its language as early as 1952. Two languages were developed under Dr. Hopper's directions : FLOWMATIC was a commercial and business language which could easily be put together from the contents of a flowchart, whereas MATHEMATIC was a mathematical language. These two languages were an early example of the development of high-level languages in different directions - business and commercial, and science and mathematics. Since then, many other high-level languages have been produced. Today there are over 200 high-level languages. However, most of these are for very special purposes or are designed to solve problems in a very specific applications area. Some of the most common high-level languages have been briefly described below. The primary objective is to provide some insight into these languages rather than to provide detailed knowledge required to write programs.

#### FORTRAN

One of the oldest and the most popular high-level language is FORTRAN which stands for *FORMula TRANslation*. Originally developed by IBM (International Business Machine Corporation) for its 704 computer in 1957, FORTRAN has undergone several revisions so that the language has been evolving into a wider and more useful language with time. (The original FORTRAN was



soon followed by FORTRAN II. The next popular and advanced version was FORTRAN IV. In order to allow a program that was written for one computer system to be run on another computer system, this version was standardized by the American National Standards Institute (ANSI) in the year 1966. FORTRAN thus has the distinction of being the first standardized language. In 1977 an updated version of FORTRAN IV, known as FORTRAN 77, was announced and standardized by ANSI. It contains several additional features which are not a part of FORTRAN IV, e.g. character and file handling, constructs related to a more structural approach to programming.

(FORTRAN was designed to solve scientific and engineering problems and is currently the most popular language among scientists and engineers. The language is oriented towards solving problems of a mathematical nature and has been designed as an algebra-based programming language. Any formula or those mathematical relationships that can be expressed algebraically can easily be expressed as a FORTRAN instruction, e.g.  $A = B + C - D$ .) To illustrate the nature of FORTRAN programs, a simple FORTRAN program to compute and print the sum of 10 numbers is given below.

```

C  FORTRAN PROGRAM TO COMPUTE
C  THE SUM OF 10 NUMBERS
SUM = 0
DO 50 I = 1, 10
  READ (5, 10) N
10  FORMAT (F6. 2)
  SUM = SUM + N
50  CONTINUE
  WRITE (6, 20) SUM
20  FORMAT (1X, 'THE SUM OF GIVEN
-  NUMBERS=', F10. 2)
  STOP
  END

```

From the example you can see that a FORTRAN program consists of a series of statements. These statements supply input/output, calculation, logic/comparison, and other basic instructions to the computer. The words READ, WRITE, DO, and STOP in the statements mean exactly what you would expect. A FORTRAN program requires that certain parts of every statement be placed in certain columns. Statement numbers, which are optional in FORTRAN, are placed in columns 1-5 (10, 50 and 20 in this example). A comment statement starts with a C in the first column (the first statement of this example). Comment statements are used in programs for the purpose of documentation or explanation designed to assist anyone reading the source program listing.

Comments do not form a part of the program logic and are ignored by the computer. The actual FORTRAN statement is placed in columns 7-72. A character in column 6 means that the statement in the previous line is being continued in this line. Columns 73-80 are ignored by the computers. The programmer may use these columns for any purpose, such as numbering each statement or writing a program code name.

In the example above, the value of SUM is first initialized to 0. The next statement tells the computer to do a loop that starts at the DO statement and ends in line having label 50, which is a CONTINUE statement. Inside the loop, values of N are read and added to SUM one by one. After the computer loops 10 times, reading and accumulating the sum of 10 numbers, the computer goes out of the loop and drops down to the next statement. This is the WRITE statement which prints the message : THE SUM OF GIVEN NUMBERS = followed by the computed value of SUM. The next statement, which is a STOP statement tells the computer to stop the execution of the program. Finally, the END statement tells the computer that there are no more instructions or statements in the program. The data for this program is contained in a separate file and is not shown in the program.

## COBOL

(COBOL is an acronym for COmmon Business Oriented Language.) As its name implies, this language was designed specifically for business data processing and today it is the most widely used business-oriented programming language. Unlike FORTRAN, which gradually developed into a full-fledged language, the vocabulary and grammar of COBOL were worked out in 1959-1960 by a committee of the COncference on DATA SYstems Languages (CODASYL) as a joint effort of computer users, manufacturers, and the United States government. After the vocabulary and grammar were defined by this committee, the various manufacturers wrote the compilers for their computers. Since 1960, the language has been revised, but revision by manufacturers have been rare, because other CODASYL committees have continued to maintain, revise, and extend the initial specifications. An ANSI COBOL standard was first published in 1968, and a later version was approved in 1974. As long as these standards are followed, a COBOL program can be run on any computer system with an ANSI COBOL compiler.

COBOL was designed to have the appearance and structure of a business report written in English. Thus, a COBOL program is constructed from sentences, paragraphs, sections, and divisions. All COBOL programs must have four divisions namely, the identification



division, the environment division, the data division, and the procedure division. The nature of COBOL program is illustrated below with the help of a simple COBOL program to compute and print the sum of given numbers.

#### IDENTIFICATION DIVISION.

PROGRAM-ID. SUMUP.

AUTHOR. P K SINHA.

\* THIS PROGRAM COMPUTES AND PRINTS

\* THE SUM OF GIVEN NUMBERS.

#### ENVIRONMENT DIVISION.

##### CONFIGURATION SECTION.

SOURCE-COMPUTER. BURROUGHS-6700.

OBJECT-COMPUTER. BURROUGHS-6700.

##### INPUT-OUTPUT SECTION.

##### FILE-CONTROL.

SELECT DATA-FILE ASSIGN TO  
CARD-READER.

SELECT OUTPUT-FILE ASSIGN TO PRINTER.

#### DATA DIVISION.

##### FILE SECTION.

##### FD DATA-FILE

RECORD CONTAINS 80 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS INPUT-DATA-RECORD.

01 INPUT-DATA-RECORD.

05 N PICTURE 9(6)V99.

05 FILLER PICTURE X(72).

FD OUTPUT-FILE

RECORD CONTAINS 132 CHARACTERS

LABEL RECORD IS OMITTED

DATA RECORD IS OUTPUT-RECORD.

01 OUTPUT-RECORD.

05 FILLER PICTURE X.

05 TITLE PICTURE X(25).

05 SUM PICTURE 9(10)V99.

05 FILLER PICTURE X(94).

##### WORKING-STORAGE SECTION.

77 MESSAGE PICTURE X(25) VALUE IS  
"THE SUM OF GIVEN NUMBERS=."

#### PROCEDURE DIVISION.

##### OPEN-FILES.

OPEN INPUT DATA-FILE.

OPEN OUTPUT OUTPUT-FILE.

##### INITIALIZATION.

MOVE SPACES TO OUTPUT-RECORD.

MOVE ZERO TO SUM.

##### PROCESS-LOOP.

READ DATA-FILE AT END GO TO

PRINT-PARA.

ADD N TO SUM

GO TO PROCESS-LOOP.

PRINT-PARA.

MOVE MESSAGE TO TITLE.

WRITE OUTPUT-RECORD.

END-OF-JOB.

CLOSE DATA-FILE.

CLOSE OUTPUT-FILE.

STOP RUN.

It can be easily seen from this example that COBOL is a self-documenting language. Self-documenting languages are those that do not require much explanation in order to be understood by some one reading the program instructions.

The self-documenting aspect of COBOL is made possible by its English like sentence and paragraph structure and the very generous maximum symbolic field-name length of 30 characters. With a field-name length of up to 30 characters, the name can clearly identify the field and its purpose. COBOL programmers should ensure that the field name that are used are meaningful, so that the self-documenting feature of the language is not lost.

Like FORTRAN, most COBOL compilers require that certain parts of every statement be placed in certain columns. There are 80 columns in a line out of which columns 1-3 are used for page number and columns 4-6 identify line numbers. The use of sequence numbers is optional and can be omitted. An asterisk (\*) in column 1 indicates a comment line and the entry is not compiled to produce object code. Comment lines are actually some notes revealing the intentions of the programmer and are used for program documentation. The actual COBOL statements are placed in columns 8-72. Notice that there are two margins: A and B. The A margin that starts at column 8 is used to start a new division, section, or paragraph. The B margin that starts at column 12 is used to start any sentence. Finally, columns 73-80, which the compiler ignores, can be used to write some identification.

## BASIC

BASIC (Beginners All-purpose Symbolic Instruction Code) was developed by Professors John Kemeny and Thomas Kurtz in the year 1964 at Dartmouth College in the United States. Their purpose was to develop a language which would be very easy to learn and thus can be used by the undergraduate students in all fields of study. The language has few grammatical rules and can be learnt in a few hours of concentrated study. In order to understand and write programs in BASIC, it is not necessary to learn complex programming techniques. (A person with little or no knowledge of computers or programming can learn to use BASIC.)



write BASIC programs in a short period of time. Because of its simplicity and bias towards the user, BASIC is even being used by school students. It is a language well suited for use in education and has become extremely popular with microcomputer users.)

Unlike FORTRAN or COBOL, BASIC is an interpreter based language. Instead of compilers, interpreters are generally used in microcomputer systems to translate BASIC instructions into machine-language code. Thus, as a BASIC program is being entered, its statements are checked for syntax errors which can be immediately corrected. This feature of BASIC makes it one of the most popular computer languages used in microcomputer systems.) It is available in almost all microcomputers and even in some pocket calculators. Though simple and easy to learn, BASIC is quite flexible and reasonably powerful. (It can be used for both business and scientific applications. Probably the greatest drawback of this language is that it has not yet been standardized. The language varies significantly from one computer system to another. Thus, a BASIC program written on one computer may not work on another unless modified.)

A Basic program to compute and print the sum of 10 numbers is given below :

```

5  REM PROGRAM TO COMPUTE
6  REM THE SUM OF 10 NUMBERS
10 LET S = 0
20 FOR I = 1 TO 10
30 READ N
40 LET S = S + N
50 NEXT I
60 PRINT "THE SUM OF GIVEN NUMBERS="; S
70 DATA 4, 20, 15, 32, 48
80 DATA 12, 3, 9, 14, 44
90 END

```

It can be observed from this example that a BASIC program is made up of a series of statements. Each statement starts with a statement number and a key word, followed in most cases by some type of action. For example, in the statement, "40 LET S = S + N", 40 is the statement number, LET is the key word, and S = S + N is the action. The first statement of the program which is a REM statement is a remark being made for the purpose of explaining or documenting a program step. It has no effect on the program logic. The instruction must have a line number, the key word REM, and any remark that the programmer wishes to make. In our example, the remark statement was used to name the program. The two DATA statements in the program are used to furnish the input data.

The FOR and NEXT statements control the loop that is meant for adding the 10 numbers one by one. The END statement stops the program execution.

## PASCAL

(Named after the famous French mathematician Blaise Pascal, this language was first introduced in the year 1971 by Professor Niklaus Wirth of the Federal Institute of Technology in Zurich Switzerland. His aim was to develop a language after the concepts associated with structured programming. Thus, PASCAL was the first language to fully embody in an organised way the concepts of structured programming. The language is relatively easy to learn, and it allows the programmer to structure the programming problem. This means that the program must be written in logical modules which are in turn called by a main controlling module. In other words, PASCAL is designed to force us to look at a problem in a logical way and to lay out a solution before we begin writing the program.) It is based on the theory that the use of too many GO TO statements in a program makes the program clumsy and unstructured. The logic of such programs becomes complex and these programs are difficult to understand and maintain because they lack any recognizable structure or flow of control. Hence, as a good programming practice, a programmer should avoid the use of GO TO statements in his program as far as practicable. The features of PASCAL help us in overcoming this problem. (The language is designed in such a way that complete PASCAL programs can be written without the use of GO TO statements. The process of looping or repeating a sequence is handled automatically by special loop control statements. PASCAL is thus widely recognised as a language that instills good programming habits in a programmer. Owing to this reason, this language is also extensively used to teach programming to beginners.) The features of PASCAL allow it to be used for both scientific and business applications. Hence, it is a very powerful language and has been implemented on several different computers including minicomputers and microcomputers.) A PASCAL version of a summing program similar to those presented earlier in other languages is shown below.

```

PROGRAM SUMNUMS (INPUT, OUTPUT);
(* PROGRAM TO COMPUTE THE SUM OF 10
NUMBERS *)
(* DECLARATION OF VARIABLES *)
VAR SUM, N : REAL;
VAR I : INTEGER;

```

```

BEGIN

```

```

  SUM := 0;
  FOR I := 1 TO 10 DO

```



```

BEGIN
  READ (N);
  SUM := SUM + N;
END;
Writeln ('THE SUM OF GIVEN NUMBERS=',
  SUM);
END.

```

The first line of the program contains the name of the program which is SUMNUMS. This is followed by two comment lines which are used for documentation purpose. Any comment can be placed within the symbols (\* and \*) to document a PASCAL program. Then, all the variables are declared. The variables SUM and N are declared as real and hence they can be assigned any real number. Similarly, the variable I, that has been declared to be an integer variable, can be assigned any integer value. The heart of the program starts with the word BEGIN and ends with the word END. First, the variable SUM is initialized to zero. The next statement starts a DO loop that reads and computes the sum of the 10 numbers. Finally, the statement having Writeln prints the result of the program. It may be observed that PASCAL programs are composed of blocks starting with BEGIN and terminating with END. All variables are declared at the beginning of the program and program statements proceed in a logical flow from start to finish.

### PL/I

PL/I stands for programming language one. It was designed in the mid-1960s by IBM as a general purpose language having features similar to COBOL for business applications and features similar to FORTRAN for scientific applications in addition to other features such as string manipulation and list processing. The intention was to create a universal language which would be adequate for programming any kind of application. A PL/I standard was produced by an ANSI committee in the year 1976. A subset of this full standard, known as PL/I-G is also available for use with small computers.

Although PL/I is one of the most versatile and the most powerful of the programming languages, it is not the most commonly used. The main reason behind this is that since PL/I has features found in both FORTRAN and COBOL, it is a sophisticated language. It is enormous by any standards and is not easy to learn in its totality. Furthermore, the complexity of the language makes a compiler and support packages for the full language quite large. Because of its size and heritage, the language so far has been available primarily on IBM equipment and is not very widely used or accepted.

A PL/I program for the summing problem is given

below to illustrate the nature of the language.

```

SUMNUMS : PROCEDURE OPTIONS (MAIN);
/* PROGRAM TO COMPUTE THE SUM OF 10
NUMBERS */
DECLARE (SUM, N) FIXED;
DECLARE I FIXED;
SUM = 0;
DO I = 1 TO 10;
  GET (N);
  SUM = SUM + N;
END;
PUT EDIT ('THE SUM OF GIVEN NUMBERS =', SUM)
(A,F5);
END;

```

It can be seen from this example, that the basic element of a PL/I program is a statement. Each statement is terminated by a semicolon (;) and several statements may be combined to form a procedure. A procedure may represent an entire small program (as in this example) or a block or module of a more complex program. Because of its modular structure, a beginner need only learn a small part of the language in order to write programs for a particular type of application. Moreover, modular procedure blocks and other features of the language allow the use of the various concepts of structured programming.

### OTHER HIGH-LEVEL LANGUAGES

The high-level programming languages that were discussed above are not necessarily the most important or most popular languages. These languages were presented in some detail to give you a better understanding of computer programming and nature of high-level languages in general. There are several other programming languages which are equally important and popular. Some of these languages are briefly discussed in this section.

**RPG.** Report program Generator (RPG) is a business-oriented, general purpose programming language. As the name implies, the language is designed to generate the output reports resulting from the processing of common business applications. The language was developed by IBM as a result of their customer requests for an easy and economic mechanism for producing reports and was launched in the year 1961 for use on the IBM 1401 computer. The latter version of RPG, called RPG II, greatly improved the language and gave it additional capabilities.

RPG is considerably different from other programming languages. Instead of writing instructions or statements, the programmer uses very detailed coding



sheets to write his specifications about input, calculations, and output. These sheets specify exactly what is to be done, and then the computer uses them to generate the necessary instructions to perform the desired application. Thus, RPG is easier to learn and use as compared to COBOL. Moreover, RPG can duplicate any COBOL program. Owing to these reasons, RPG is commonly used on many small computers and in small businesses. It is well suited for applications where large files are read, few calculations are performed, and output reports are created. However, RPG has restricted mathematical capability and cannot be used for scientific applications.

**ALGOL.** Like FORTRAN, ALGOL (ALGO<sup>r</sup>ithmic Language) is also one of the earliest and the most influential high-level languages that was designed for scientific applications. It was designed by an international group of mathematicians, and developed by groups in Europe and the United States. The language was first introduced in 1958, resulting in ALGOL58. Later on, it was revised in 1960 and this revised version, known as ALGOL60, was the most widely used version of the language. The most recent, and the most powerful, version is ALGOL68. Like PASCAL and PL/I, ALGOL is a block-structured or modular language that is well suited for use in a structured programming setting. One area in which ALGOL, as originally defined, is quite deficient is that of input/output. The language was primarily designed as a way of expressing algorithms, and I/O statements were not made a part of the official language. ALGOL has not proven to be as popular in the United States as in Europe for practical work, although it is widely used in universities, especially in computer science departments because of its elegance and power. The lack of I/O facilities, plus the fact that the largest computer manufacturer, IBM, did not favour ALGOL early in its existence, has contributed to its lack of widespread usage.

**APL.** APL (A Programming Language) was developed by Kenneth in 1962. It is a very powerful programming language that is well suited for specifying complex algorithms. Much of its power is vested in the rich set of mathematical operators available, enabling easy manipulation of matrices and arrays of highest rank. This language is a real-time language developed primarily for scientific applications. It is usually used in an interpretive and interactive manner - an environment which greatly enhances its power. Since APL uses a rather large and unusual character set, a special keyboard and terminal is necessary for its implementation. Until recently, only certain IBM equipment supported APL. However, the advent of low-cost terminals capable of handling many different type of fonts and the growing number of APL users have brought it into wider, but still limited, usage.

**ADA.** ADA is a new general purpose programming language. It was developed in the year 1980 at the request of the U.S. Department of Defence (DOD) for use in military applications. The language was developed at Honeywell Computer Company by a group of scientists headed by Ichbiah. It is named in honour of Lord Byron's daughter Ada Augustha Lovelace. Ada was a close friend of Charles Babbage, who has made significant contributions in some of the earlier developments of computer systems. Ada used to work with Babbage. She is considered by many to be the first "programmer" because she wrote the first computer program for the mechanical computer (Analytical Engine) developed by Charles Babbage.

Ada is an extremely complicated language with a very large number of features. In addition to the normal types of statements and commands, ADA also allows the use of packages. A package allows for a collection of related computational procedures and resources. Packages are specified or written into the declarative part of the program, which is typically at the top of an ADA program. Then various procedural statements in the program can access the package and use it. Another important feature of ADA is the use of tasks. Tasks are used to allow concurrent programming which is very useful for military applications. Like packages, tasks have a specification part. Once specified, tasks can be used within the body of the program as and when needed. It is expected that most future programs written for the Department of Defense, U.S.A. will have to be in this language. As a result, this new language may become very popular in the field of defense.

**LISP.** LISP stands for LISt Processing. This language was designed by McCarthy and is suitable for nonnumeric applications. It is a powerful language for handling logical operations. Because of this feature, the language is extensively used in the areas of pattern recognition, artificial intelligence, and for simulation of games.

**SNOBOL.** SNOBOL (StriNg Oriented SymBolic Language) is another language used for nonnumeric applications. As the name implies, the language was basically designed to manipulate strings of characters. SNOBOL has powerful string manipulation features that facilitate various types of operations on strings of characters such as string comparison, splitting of a string, combining two strings, etc. Thus, this language has been widely accepted for applications in the area of text processing.

**C.** C is a relatively new language which is becoming very popular day-by-day. It was designed at Bell Telephone Laboratories, U.S.A. Like PASCAL and ALGOL, C is a block structured language and has several features that



allow the use of the various concepts of structured programming. Moreover, a special feature of this language, that is normally absent in other high-level languages, is that it allows the manipulation of internal processor registers of the computer. Thus, the language also enjoys the advantage of having some of the powers of assembly language. Because of this feature, C language is now being extensively used for systems programming like design of compilers and operating systems. Most computer vendors of today, supply this language along with their computer systems.

**PROLOG.** PROLOG stands for *PRO*gramming in *LOG*ic. It is a very new programming language designed for handling complex logical operations. The language is being used to design intelligent computer systems and is expected to gain popularity in the near future.

## CHARACTERISTICS OF A GOOD LANGUAGE

In the previous section, we have seen that there are some high-level languages which are very popular and there are others which are used only by a small group of programmers. Why do programmers prefer one language over another? One obvious reason is the area of application. However, another equally important reason is the characteristics of the language. Several properties believed to be important with respect to making a language good and usable by human beings are briefly outlined below.

1. **Simplicity.** Programming languages that are simple and easy to learn and use are liked by many programmers. For example, BASIC is used by many programmers only because of its simplicity. Thus, a language should provide a programmer with a clear, simple, and unified set of concepts which can be easily grasped. There should be a minimum number of different concepts, with the rules for their combination being as simple and regular as possible. However, the power needed for the language should not be sacrificed for simplicity.
2. **Naturalness.** A language should be natural for the application area it has been designed. In other words, it should be problem-oriented. It should provide appropriate operators, data structures, control structures, and a natural syntax in order to facilitate the users to code their problem easily and efficiently. Often if a major amount of programming in a particular area is required, it is extremely useful to develop a programming language just for that class of

applications. FORTRAN and COBOL are good examples of scientific and business languages respectively that possess high degree of naturalness.

3. **Efficiency.** Efficiency is certainly a major element in the evaluation of any programming language. Thus, while designing a compiler or an interpreter for a particular language, system programmers must give due consideration to space and time efficiency. A programming language should be such that its programs are efficiently translated into machine code, are efficiently executed, and acquire as little space in the memory as possible.
4. **Structuredness.** Structuredness means that the language should have necessary features to allow its users to write their programs based on the concepts of structured programming. The main reason behind this is that, this property of a language greatly affects the ease with which a program may be written, tested, and maintained. Moreover, it forces a programmer to look at his problem in a logical way and hence he creates fewer errors while writing a program for his problem. PASCAL, PL/I, and ALGOL are some of the languages having this property.
5. **Compactness.** Users of a high-level, problem-oriented language should be able to express intended operations concisely, since this is one of the fundamental reasons for having it. A verbose language can tax the programmer's sheer writing stamina and thus reduce its usefulness. COBOL is generally not liked by many programmers because it is verbose in nature and lacks compactness.
6. **Locality.** A programming language should be such that while writing a program, a programmer need not jump around visually as the text of the program is prepared. COBOL lacks locality because data definitions are separated from processing statements, perhaps by many pages of code. On the other hand, APL is very local since it requires no declarations, thus permitting the programmer to concentrate almost solely on the part of the program around the statement currently being worked with.
7. **Extensibility.** A good programming language should also allow extension through simple, natural, and elegant mechanisms. Almost all



languages provide subprogram definition mechanisms for this purpose, but there are some languages that are rather weak in this aspect.

easily be written and used as and when we need them.

8. *Suitability to Environment.* Depending upon the type of application for which a programming language has been designed, the language must also be made suitable to its environment. For example, a language designed for real time applications must be interactive in nature. On the other hand, languages used for data processing jobs like pay-roll, stores accounting, etc., may be designed to be operative in batch mode. A language designed for interaction, say APL, is not as usable in batch mode. Likewise, a language designed for batch usage may prove quite frustrating if used interactively from a terminal.

## SUBROUTINES

As we are going to abandon the chapter on program writing, it is felt essential to introduce the concept of subroutines which are very frequently used in programming.

Suppose we are writing a program for solving a trigonometric problem. During the problem, suppose we need to calculate the square root of a number three times. Should we write the square root procedure into our program thrice? Why can we not write the square root procedure just once, and use it every time we need it?

We could of course write the steps required for the calculation of square root each time we need them in the program. But a much easier way of handling the problem is to write these steps once and then refer to them each time we need them.

*Subroutines* (also called subprograms) are programs written in such a way that they can be brought into use in other programs and used whenever needed, without rewriting. In other words, a subroutine is any standardized program written in such a way that it can be used as part of another program whenever necessary. A subroutine is normally invoked through other programs by the use of CALL statements.

There are many subroutines such as those for finding square roots, sines, cosines, logarithms, etc. which are used over and over by many programmers. Such subroutines are usually supplied by the computer manufacturers along with the language compiler and are referred to as *built-in functions*. Other subroutines can

## QUESTIONS

1. Discuss the analogy between a computer language and a natural language.
2. How does a computer language differ from a natural language?
3. Name the three different categories of computer languages.
4. What is a machine language? Why is it required?
5. What are the advantages and limitations of machine language?
6. When is a computer language called machine dependent? What is the main disadvantage of such a language?
7. A machine language instruction has a two-part format. Identify these parts and discuss the function of each.
8. What is a mnemonic? How is it helpful in case of computer languages?
9. What is an assembly language? What are its advantages over machine language?
10. What is an assembler?
11. What is the difference between a source program and an object program?
12. What is a macro instruction? How does it help in reducing a programmer's job?
13. What are high-level languages? Why are they known as problem-oriented languages? Name some high-level languages.
14. Why are high-level languages easier to use?
15. What is a compiler? Why is it required? A computer supports five high-level languages? How many compilers will this computer have?
16. Illustrate the machine independent characteristic of high-level languages.



17. What type of errors can be detected by a compiler ? What type of errors cannot be detected ?
18. What is an interpreter ? How does it differ from a compiler ?
19. List out the characteristics necessary for a programming language to be considered as a high-level language.
20. What are the advantages and limitations of high-level languages ?
21. Briefly describe the development and use of the following programming languages :
  - (a) FORTRAN
  - (b) COBOL
  - (c) BASIC
  - (d) PASCAL
  - (e) PL/1
  - (f) ADA
  - (g) RPG
  - (h) ALGOL
  - (i) APL
  - (j) SNOBOL
  - (k) LISP
22. What characteristics are desirable for a good computer language ?
23. What is a subroutine ? How do subroutines help in program writing ?
24. Would you be equally likely to choose FORTRAN or COBOL for a given task ? Why ?
25. What is the purpose of a language processor ? Are language processors hardware or software ?
26. While writing a program, a programmer erroneously instructed the computer to calculate the area of a rectangle by adding the width to its length (that is,  $AREA = LENGTH + WIDTH$ ) instead of multiplying the length and width. Would you expect the language processor to detect this error ? Explain.
27. A programmer eliminates all language processor errors from his program and then runs it to get printed results. The programmer therefore concludes that the program is complete. Comment.
28. It is said that an assembly language is "one-for-one" but a high-level language is "many-for-one". Explain what this means.
29. What is meant by standardization of a language ? Why is it important ?
30. What is the role of comments in a program, and how are they treated by the language processor ?
31. What is a self-documenting language ? Illustrate with an example.
32. List out some of the program preparation techniques that are often included under the term "structured programming".
33. Give the full form of the following terms :
  - (a) FORTRAN
  - (b) COBOL
  - (c) BASIC
  - (d) PL/1
  - (e) RPG
  - (f) ALGOL
  - (g) APL
  - (h) LISP
  - (i) SNOBOL



CHAPTER 13

# 13. SYSTEM IMPLEMENTATION AND OPERATION

In the previous two chapters, we have discussed the analysis, design, and coding phases of a computerised system. After the computer programs have been prepared, the computer information system enters the implementation and operation phase. The goal of this chapter is to describe the principal activities of the implementation and operation phase, which relate to testing and debugging of programs, complete documentation of the system, changeover to the new system, and system modification and maintenance.

## TESTING AND DEBUGGING

So long as computers are programmed by human beings, computer programs will be subject to errors. Program errors are known as *bugs* and the process of detecting and correcting these errors is called *debugging*. In

general, *testing* is the process of making sure that the program performs the intended task, and *debugging* is the process of locating and eliminating program errors. Testing and debugging are vital steps in developing computer programs. They are also time-consuming steps. In fact, the time spent in testing and debugging often equals or exceeds the time spent in program coding.

In general, there are two types of errors that occur in a computer program - *syntax errors* and *logical errors*. Syntax errors result when the rules or syntax of the programming language are not followed. Such program errors typically involve incorrect punctuation, incorrect word sequence, undefined terms, or misuse of terms. For example, the FORTRAN statement  $C = (A + B/2$  has a syntax error. In this example, the problem is a missing closing parenthesis which should be placed in the appropriate place depending on the logic of the program.



All syntax errors must be found and corrected before there is any chance of running the program. Almost all language processors are designed to detect syntax errors. The language processors print error messages on the source listing that indicate the number of the statement with errors and give hints as to the nature of the error. These error messages are very useful and are used by the programmers to rectify all syntax errors in their programs. Thus, it is a relatively easy task to detect and correct syntax errors.

It should be noted that in high-level languages such as FORTRAN and COBOL a single error often causes multiple error messages to be generated. There are two reasons for multiple error messages. One is that high-level language instructions often require multiple machine steps. The other reason is that symbolic instructions are often dependent upon other instructions and if an instruction containing an error is one that defines a field name, all instructions in the program using that field name will be listed as errors. The error message will say that a field being used is not a defined name. In such a case, removal of the single error will result in the removal of all associated error messages.

The second type of error, a *logical error*, is an error in planning the program's logic. In this case, the language processor successfully translates the source code into machine code. The computer actually does not know that an error has been made. It follows the program instructions and outputs the results, but the output is not correct. The problem is that the logic being followed does not produce the results that were desired. When a logical error occurs, all you know is that the computer is not printing the correct output. The computer does not tell you what is wrong. For example, if a FORTRAN instruction should be "A = B\*C" but has been coded as "A = B+C", this error will not be detected by the language processor since no language rules have been broken. However, the output produced will not be correct. Thus, it is an example of a logical error.

In order to determine whether or not a logical error exists, the program must be tested. The purpose of testing is to determine whether the results are correct. The testing procedure involves running the program to process input test data that will produce known results. By running the program and comparing the obtained answers to the known correct results, the accuracy of the program logic can be determined. Logic errors are typically due either to missing logic or to incorrect logic. If the logic is wrong, the answers generated from the test data will be wrong. These errors are the easiest of the logic errors to find. Errors caused by missing logic result from logical situations that the program was not designed to handle. As an example, suppose that a numeric field is to be used in an arithmetic process and that

the data entry operator enters a value for the field that is not numeric. The program logic should determine that the data are not numeric prior to attempting the arithmetic process. If that logic is missing and nonnumeric data is used in an arithmetic operation, the program will fail. This type of bug can be difficult to find. The only way for this error to occur is for nonnumeric data to be entered into a numeric field. It is possible for the program to be used for weeks, months, or years before this happens and the error in program logic shows up.

In order to completely test the program logic, the test data must test each logical function of the program. Hence, the selection of proper test data is important in program testing. In general, the test data selected for testing a program should include :

1. Normal data, which will test the generally used program paths.
2. Unusual but valid data, which will test the program paths used to handle exceptions. Such data might be encountered occasionally in running the program.
3. Incorrect, incomplete, or inappropriate data, which will test the error-handling capabilities of the program. This is done to see how the program reacts in abnormal and unusual circumstances. Good programs should be able to handle abnormal data without blowing up or generating meaningless output.

If a program runs successfully with the test data and produces correct results, it is normally released for use. However, even at this stage errors may still remain in the program. In case of a complex problem, there may be thousands of different possible paths through the program and it may not be practical or even possible to trace through all these paths during testing. There are certain errors in complex systems that remain dormant for months and years together and hence it is impossible to certify that very complex systems are error-free.

There are several ways to locate and correct logical errors some of which are briefly described below :

1. One approach is to study the source code producing the incorrect results and try to determine the cause of problem. However, some errors are difficult to find, and simple inspection of the source program does not reveal the nature of the error.



2. Another approach is to put several print or write statements in the program that indicate the values of intermediate computations. These statements can also be used to tell you what is happening during the execution of the program. Once the errors have been found and corrected, these print or write statements are removed from the program.
3. The third approach involves the use of tracing routines which are software tools provided to the programmer to help in debugging the program logic. *Tracing routines* or *debug packages* assist the programmer in following the logic by printing out intermediate calculation results and field values that are used in making logical decisions in the program. Using these techniques, the programmer can follow the program's execution step by step in order to determine where the logic is in error.
4. In a few cases, the logical error can be so difficult to find, and the number of fields involved are so numerous, that the only way to uncover the source of the error is to look at a printout of the contents of memory. This printout is called a *storage dump* or *memory dump*. The memory dump lists the instructions and data held in the computer's memory in their raw form, that is, their binary or equivalent hexadecimal or octal form. The programmer can then study this listing for possible clues to the cause of the programming error(s). Most programmers of high level languages resort to the use of memory dumps only when all other methods of detecting the logical error fails.

2. It is a positive activity that seeks to demonstrate that the program is correct and does, in fact, meets its design specifications.

3. Testing is complete when all desired verifications against specifications have been performed.

4. Testing, can and should be planned. It is a definable task in which the how and what to test can be specified. Testing can be scheduled to take place at a specific time in the development cycle.

5. Testing can begin in the early stages of the development effort. Of course the tests themselves must be run near the end of a project, but the decisions of what to test, how to test, with what kind of data, can and should be completed before the coding is started.

It is a negative activity in that it is centered around elimination of known errors or bugs.

Debugging is finished when there are no known errors. However, debugging is a process that ends only temporarily, because subsequent execution of a program may uncover other errors - thereby restarting the debugging process.

Because debugging is a reactive procedure which stems from testing, it cannot be planned ahead of time. The best that can be done is to establish guidelines of how to debug and develop a list of "what to look for."

Debugging, on the other hand, cannot begin until the end of the development cycle, because it requires an executable program.

### Difference Between Testing and Debugging

Testing and debugging are two separate tasks. They should not be confused with each other. The differences between these two processes are outlined below :

#### TESTING

1. Testing is a process in which a program is validated.

#### DEBUGGING

Debugging is a process in which program errors are removed.

### DOCUMENTATION

A computerised system cannot be considered to be complete until it is properly documented. In fact, documentation is an on-going process that starts in the study phase of the system and continues till its implementation and operation phase. Moreover, documentation is a process that never ends throughout the life of the system. It has to be carried out from time to time as and when the system is modified during its maintenance phase.



System documentation involves collecting, organizing, storing, and otherwise maintaining a complete historical record of programs and other documents used or prepared during the different phases of the system. Proper documentation of a system is necessary due to the following reasons :

1. It solves the problem of indispensability of an individual for an organisation. In large organisations, over the years, the designer and developer of a software system may not be in the same organisation. In such cases, even if the person who has designed or programmed the system, leaves the organisation, the documented knowledge remains with the organisation which can be used for the continuity of the system.
2. Maintainability of computer software poses a concern to major data processing installations. The key to maintenance is proper and dynamic documentation. A well documented system is easy to modify and maintain in the future. It is easier to understand the logic of a program from the documented records rather than its code. System flowcharts, program flowcharts, or comments used within the programs prove to be very helpful in this regard.
3. Documented records are quite helpful in restarting a project that was postponed due to some reason or the other. The job need not be started from scratch and the old ideas may still be easily recapitulated which saves lot of time and avoids duplication of work.

Documentation of a software system is normally provided in the following forms :

1. *Comments.* Comments are very useful aid in documenting a program. From maintenance point of view, comments have been considered to be a must. Comments are used within a program to assist anyone reading the source program listing. They are used to explain the logic of the program. They do not contain any program logic and are ignored (not translated) by the language processor.

Comments should be used intelligently to improve the quality and understandability of the program. They should not be redundant, incorrect, incomplete, or written in such a way that cannot be understood by anyone else. For example, a redundant comment for the statement N

N+1 would be "INCREMENT N BY 1". Useful comments are those which describe the meaning of a group of statements such as "READ AND ECHO PRINT THE INPUT DATA". In other words, comments should mediate between the program and the problem domain. Almost all high level languages provide the facility of writing comments along with the source code of a program and it is suggested that programmers should liberally use this facility for proper documentation of their programs.

2. *Systems Manual.* A good software system must be supported with a standard systems manual that contains the following information :

- a. A statement of the problem clearly defining the objectives of the computerised system and its usefulness for various categories of users.
- b. A broad description of the system specifying the scope of the problem, the environment in which the programs function, the system limitations, the form and type of input data to be used, and the form and type of output required.
- c. Specific program names along with their description and purpose.
- d. Detailed system flow charts and program flow charts cross referenced to the program listing.
- e. Narrative description of the program listings, the processing and calculations performed and the control procedures.
- f. A source listing of all the programs together with full details of any modifications made since.
- g. Description and specifications of all input and output media required for the operation of various programs.
- h. Specimen of all input forms and printed outputs.
- i. File layout, that is, the detailed layout of input and output records.



- j. The structure and description of test data and test results, storage dumps, trace program printouts, etc., used to debug the programs.

3. *Operation Manual.* A good software package must be supported with a good operation manual to ensure the smooth running of the package. It is the operator who will perform the regular processing after the system gets stabilised and not the programmer who has developed the package. Hence, the operation manual must contain the following information :

- a. Set up and operational details of each program.
- b. List of computer switches, their location, setting and purpose.
- c. Loading and unloading procedures
- d. Starting, running, and terminating procedures.
- e. A description and example of any control statements that may be used.
- f. All console commands along with errors and console messages that could arise, their meaning, reply and/or operation action.
- g. List of error conditions with explanations for their re-entry into the system.
- h. List of programs to be executed before and after execution of each program.
- i. Special checks if any and security measures, etc.

The importance of program documentation cannot be over emphasized. There have been too many problems in the past with poorly documented systems. The result is usually errors and problems with the computer programs at a later date. It is very difficult to incorporate modifications in such systems and hence they are very expensive to maintain. Owing to these reasons, several computer installations develop strict documentation standards. These

standards describe in detail how documentation is to be performed and what reports and outputs are necessary for documentation to be completed successfully.

### CHANGEOVER TO THE NEW SYSTEM

Once the programs are tested and appear to be producing correct results, the system conversion and changeover begins. At this stage, the old system, if any, is phased out and the new system is phased in. The changeover process normally involves the following operations :

- 1. Training of personnel. Everyone who will be affected by the new system should receive some training to become familiar with the changes. This training should be both system training and user training. The overall purpose of system training is to train members of the data processing department on various technical aspects of the new system. On the other hand, user training is designed to allow managers, decision makers, and other users to become familiar with the new system so as to increase their involvement and participation in the new system.
- 2. Replacement of old forms and operation procedures by new ones.
- 3. Retirement of old input and output devices in favour of new hardware.
- 4. Incorporation of necessary changes in manual methods and assignment of new jobs to various personnel.
- 5. It is also necessary to convert all the data files from manual to computer files. That is, current files are changed into a form acceptable to the processor. Files should be consolidated and duplicate records eliminated. File inconsistencies or any errors in current files must be detected and removed.

There are three different methods normally followed to carry out the changeover process. These are immediate changeover, parallel run and phased conversion. Each of these methods has its own advantages and disadvantages.



No single method is suitable for converting all types of systems and the choice of a particular method largely depends upon the prevailing conversion circumstances. These methods are briefly described below :

### IMMEDIATE CHANGEOVER

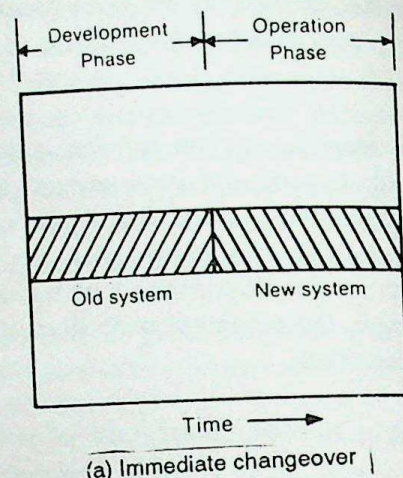
As illustrated in Figure 13.1(a), in this method of conversion, the operation of the old system is totally abandoned from the date of changeover and the complete new system starts operating from that day onwards. This method of changeover is very risky because it has been found that most systems pose some problem or the other during the changeover process. The inbetween failure of the new system due to any such problem may prove to be very harmful causing total breakdown, and the work cannot progress at all because the operation of the old system has already been abandoned. However, this method is preferred in situations where changeover time is very less, available manpower is also less, and the system is not a very critical one to business operations so that changeover problems would not trigger a disaster.

### PARALLEL RUN

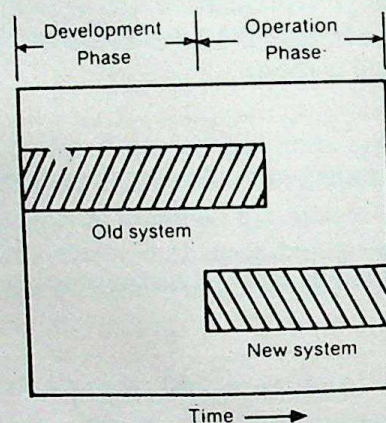
One of the best ways to implement a new system is called running in parallel. As shown in Figure 13.1(b), in this method, the old system is operated with the same data as the new system and on the same time schedule for the initial three or four cycles.

Confidence in the adequacy of the new system is normally established by comparing the data it produces with the data produced by the old system during the time of parallel operation. Some discrepancies may be discovered. Often these are due to inaccuracies in the old system that were not recognized before as inaccuracies. Unforeseen exceptions will appear for which no programming was provided. Some of the discrepancies that appear during the parallel operation stage will be due to oversights and mistakes in the programming itself. These must be corrected by further debugging before the conversion is complete.

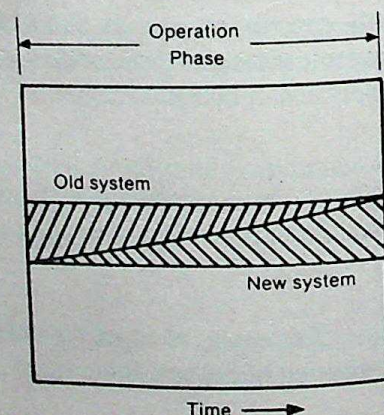
The main advantage of parallel run is the availability of old system as a backup. If there are any problems with the new programs, they can be corrected while the existing system is still being used. Thus, there is no interruption of service if there are problems with the new programs. After the bugs are removed, the new system is slowly phased in, while the old system is slowly phased out.



(a) Immediate changeover



(b) Parallel run



(c) Phased conversion

Figure 13.1. Methods of system changeover to the new system

This method, however, is not preferred in some cases because it is very expensive and time consuming. Additional manpower, which is a scarce resource in most organizations, must be provided for the operation of two



systems in parallel. Due to the requirement of additional manpower and equipment resources for parallel run, the organization is under considerable strain during the period of parallel operation, and, if it is long continued, organizational breakdowns tend to occur. To hold costs in line, experience indicates that parallel operation must not be carried on any longer than needed to establish confidence in the new system. Continuing them too long is a sign of weakness in the new system. Moreover, parallel run method of system conversion is also not used in situations where the new system differs to a great extent from the old system in the functions that it performs and its input and output.

### PHASED CONVERSION

As depicted in Figure 13.1(c), in this method, the complete changeover to the new system takes place incrementally over a period of time. The new system is gradually implemented part by part and the old system is gradually phased out. The results produced by each part of the new system is compared against the results of the old system. Any discrepancies or errors found are checked and removed. Once confidence is developed in a particular part of the new system, that part of the new system is phased in and the corresponding part (operations) of the old system is phased out. This approach is continued for each and every part of the new system. Thus, over a period of time, the new system is slowly-slowly phased in, while the old system is slowly-slowly phased out.

This method of changeover enjoys several advantages. It is not as expensive as the parallel run method because the changeover process being gradual can usually be handled with existing manpower and equipment resources. There is no danger of interruption of service if there are problems with the new programs because the corresponding part of the old system is still in operation. Moreover, the users get sufficient time to become acquainted with the new system. Hence, they can confidently handle the new system when the complete system is handed over to them. However, the phased conversion method cannot be used in situations where the time period supplied for conversion process is very less or when the new system significantly differs from the old system.

### SYSTEM EVALUATION

Once the new system is implemented and in operation, it is necessary to evaluate the system to verify whether or not it is meeting its performance objectives.

These performance objectives of the system are clearly stated during its study phase. The post implementation system evaluation is normally carried out by people who have an independent view point and are not responsible for the development and maintenance of the system. While evaluating a system, the following points are considered :

1. *Evaluation of efficiency*: The new system is compared against the old system to evaluate its efficiency in comparison to the old one. In case of any slack, the reason is analysed and if possible, necessary modifications are incorporated in the system to rectify it.

2. *Cost/benefit analysis*: The key to evaluating a system is the costs and benefits to be derived from alternate systems. This can be a long and expensive process. Information can be assembled on the actual cost of using the new system, and this cost can be compared with the anticipated cost as outlined in the report of the feasibility study group. If discrepancies are found on the high side, action can be taken to find out the reasons and then to correct or offset the causes.

Of course, benefits are not all tangible and measurable. However, those that are measurable should be expressed numerically. As a general rule, whenever we identify a tangible specific objective, we also should state the measurement that we will make in the operation phase to determine whether or not the system meets the objective. Mathematical measurements can also be made of the cost savings made possible by a computerised system, and hardware and software operating expenses can be monitored continually.

3. *Time schedule*: It should also be evaluated whether the various time schedules prepared during the study phase in the beginning have been met or not. This type of evaluation is quite helpful in preparing time schedules for the new systems that will be designed in future. Moreover, it should also be evaluated whether the planned processing procedures are being followed or not. Are all new procedures being processed on the computer? Have all old procedures been eliminated? If not, why not?



4. *Users satisfaction*: People are the final evaluators of information systems. Hence, it should be found out whether the users are satisfied with the new system or not. How useful is the system for them? How enthusiastic are they about the service they receive? Do they receive outputs in time to take necessary action? The morale of employees using or affected by a system is a good measure of the success of the project.
5. *Ease of modification*: Sooner or later, all systems must change in response to changes in their environment. New laws, changes in technology, and changes in the goals and objectives of the business are examples of causes of change. Thus, the ease with which a system can be modified to react to changes is also a significant measure of its effectiveness in achieving its objectives.
6. *Error rate*: The frequency of failure of the system should also be evaluated. This should be reduced to minimum.

## SYSTEM MAINTENANCE

No matter how good the new system is, how well it was installed and how well it may be operating, changes in business operations will force changes in the system. Changing business conditions, revised user needs, new laws, changes in technology, etc. are some factors which require that production programs be continually maintained and modified. The major cause of program maintenance is due to user requests, normally for program enhancements. As a manager uses a computer program, there is a tendency to demand additional reports and outputs from the program. Changes in data storage and organization, program bugs, and emergency program repairs are other important causes for maintenance. The remaining program maintenance is due to hardware changes, system software changes, enhancing program documentation, and scheduled and routine debugging.

Several studies have shown that, on an average, application programmers and system analysis personnel spend over half their time on program maintenance. Hence, program maintenance is an important duty of programmers and may involve all steps from problem definition through analysis, design, and program preparation. In some installations there are programmers who do nothing but maintain production programs. In fact, in many organizations well over half the total programming effort is

spent on maintenance. And it is estimated that over the life cycle of a typical application, the maintenance and enhancement costs that are incurred may be two to four times larger than the initial development costs.

Frequent change is disruptive and disturbing. Therefore, some control over changes is required. One method of achieving this control is to have all requests for change evaluated by a change control board. This board should be made up of the principal users of the system, a system analyst, and data processing personnel who are familiar with the system. Normal maintenance operations need not be approved by the change control board, but these operations should be recorded and summarized for periodic reporting to the board. Examples of maintenance activities are modifying the format of a report or rewriting a part of a computer program component to improve its efficiency. Major changes are those that significantly alter the system or require extensive personnel, hardware, or software. An example of a major change would be conversion of the system from batch processing to online terminals.

When programs are modified, it is important to make sure that program documentation is also changed accordingly. Without the existence of proper documentation that is consistent with the programs, future changes would be very difficult and costly to accomplish.

## QUESTIONS

1. What are the two types of errors that can occur in a computer program? Give an example of each to illustrate their nature.
2. How are syntax errors detected and corrected?
3. How are logical errors detected and corrected?
4. Is it easier to detect a syntax error or a logical error? Give reasons for your answer.
5. Why should a program be tested?
6. What are the different types of test data that should be selected for testing a program?
7. Why is it not possible for a very complex system



to certify that it is error free ?

in the system changeover process ?

8. What is a memory dump ? How is it useful for a programmer ?
9. Differentiate between testing and debugging.
10. Why is system documentation necessary ?
11. Discuss the different types of system documentation normally used for documenting a system.
12. What type of operations are normally carried out
13. Discuss the three different methods of system changeover along with their advantages and disadvantages.
14. What are the various factors that should be evaluated during the system evaluation process ?
15. Why is system maintenance required ? Why is it considered an important process ?
16. How can frequent program modifications be controlled ?







## CHAPTER 14

# 14. OPERATING SYSTEMS

In the last few chapters we have dealt with the planning, coding, operation, and maintenance of software systems. In this chapter, you will learn about a very important and special type of software that falls under the category of systems software. This systems software is known as operating system. The goal of this chapter is to introduce the concepts related to operating systems and to show how this particular systems software is used to make the computer a useful, easy-to-use tool.

### DEFINITION AND FUNCTIONS

An *operating system* (OS) is an integrated set of programs that is used to manage the various resources and overall operations of a computer system. It is designed to support the activities of a computer installation. Its prime objective is to improve the performance and efficiency of a computer system and increase facility, the *ease* with which a system can be used. Thus, like a manager of a company, an operating system is responsible for the smooth and efficient operation of the entire computer system. Moreover, it makes the computer system *user friendly*. That is, it makes it easier for people to interface with and make use of the computer.

Operating systems go by many different names, depending on the manufacturer of the computer. Other terms used to describe the operating system are *monitor*, *executive*, *supervisor*, *controller*, and *master control programs*. No matter by which name they are called, today most operating systems perform the following functions :

1. Processor management, that is, assignment of processors to different tasks being performed by the computer system.
2. Memory management, that is, allocation of main memory and other storage areas to the system programs as well as user programs and data.
3. Input/Output management, that is coordination and assignment of the different input and output devices while one or more programs are being executed.
4. File management, that is, the storage of files on various storage devices and the transfer of these files from one storage device to another. It also allows all files to be easily changed and



modified through the use of text editors or some other file manipulation routines.

5. Establishment and enforcement of a job priority system. That is, it determines and maintains the order in which jobs are to be executed in the computer system.
6. Automatic transition from job to job as directed by special control statements.
7. Interpretation of commands and instructions.
8. Coordination and assignment of compilers, assemblers, utility programs, and other software to the various users of the computer system.
9. Establishment of data security and integrity. That is, it keeps different programs and data in such a manner that they do not interfere with each other. Moreover, it also protects itself from being destroyed by any user.
10. Production of dumps, traces, error messages, and other debugging and error-detecting aids.
11. Maintenance of internal time clock and log of system usage for all users.
12. Facilitates easy communication between the computer system and the computer (human) operator.

An operating system performs a wide variety of jobs. Each of these jobs are performed by one or more computer programs and all these computer programs are jointly known as an operating system. Out of the complete operating system, normally, one control program resides in the main memory of the computer system. This control program is known as the *resident program* or the *resident routine*. The other programs are stored on the disk and are called *transient programs* or *transient routines*. These programs include utility programs, compilers, assemblers, etc. The control program transfers these programs into the main memory and executes them as and when they are needed. It may be recalled here that the capacity of the main memory of any computer system is very small as compared to its secondary storage devices like disks. This is because main memory is very expensive as compared to secondary storage devices. This is the reason why only the control program is stored in main memory and the rest of the operating system is stored on disks.

In effect, besides the hardware, each computer system consists of an operating system that enables a user to effectively use the system. Thus, as shown in Figure 14.1, the OS tends to isolate the hardware from the user. The user communicates with the OS, supplies application programs and input data, and receives output results. However, it is interesting to know that all the tasks performed by the OS are performed automatically. The functions of the OS are transparent to the user - he really is not at all concerned with what the OS is doing or how the OS directs the hardware to handle certain tasks.

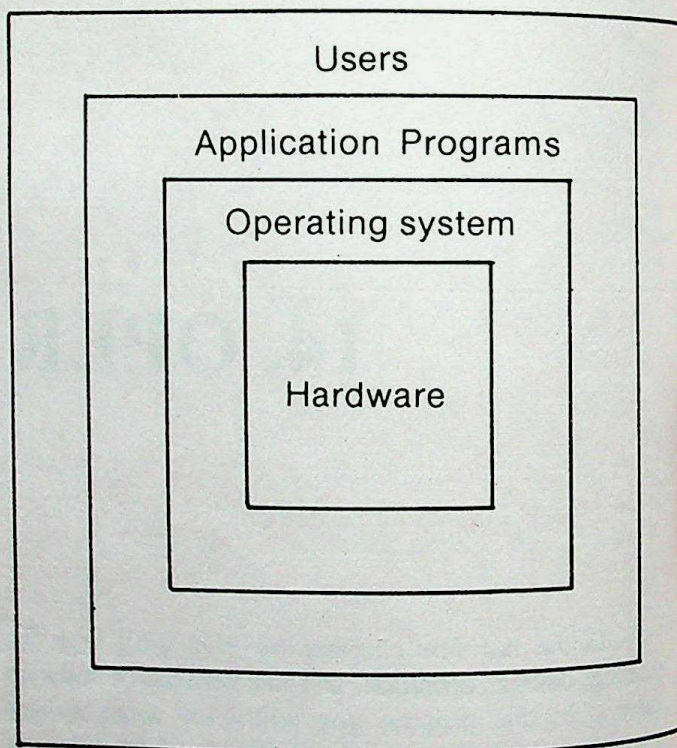


Figure 14.1. The in-between software layers isolate the hardware of a computer system from its users.

The efficiency of an operating system and the overall performance of a computer installation is judged by a combination of two main factors. They are :

1. **Throughput.** It is the total volume of work performed by the system over a given period of time.
2. **Turnaround Time.** It is also known as response time and is defined as the interval between the time a user submits his job to the system for processing and the time he receives results. Response time is especially important where



many different users share the use of the system and the overall progress of their work depends upon their receiving prompt results from the system.

## EVOLUTION OF OPERATING SYSTEMS

It is believed that one of the first operating systems was developed in the early 1950s for the IBM 701 computers. This OS was elementary in nature and was not so powerful as the operating systems of today's computers. Since then, lot of research work has been carried out in this direction with the result that today we have very powerful operating systems which are machine independent and can execute several jobs at a time on the same machine. The main aim of all the researchers involved in the development of OS was to devise ways to minimise the idle time of the computer system and to use the computer system in the most efficient and economical way.

In the early days of computers, job-to-job transition was not automatic. For each and every job to be executed by the computer, the operator had to clear the main memory to remove any data remaining from the previous job, load the program and data of the current job from the input devices, set the appropriate switches, and finally run the job to obtain the results from the output devices. After the completion of one job, the same process had to be repeated for the next job by the computer operator. Because of the manual transition from one job to another, lot of computer time was wasted since the computer remained idle while the operator loaded and unloaded jobs. In order to reduce this idle time, a method of automatic job-to-job transition was devised. With this facility, when one job is finished, the system control is automatically transferred back to the operating system which automatically performs the housekeeping jobs needed to load and run the next job.

The automatic job-to-job transition facility provided by the OS reduced the idle time of the computer to a great extent. But still there was another scope for reducing the idle time of the CPU. If you remember properly, we have seen in Chapter 9 that the speed of CPU is much more as compared to the speed of I/O devices. Hence, the CPU was normally idle while a particular job was busy with some I/O operations. So the next attempt by OS developers was to overcome this speed mis-match by executing more than one program at the same time. In this method, while one program was busy with some I/O operation, the CPU time was utilised for processing another job.

In a similar manner, there have been many improvements in the operating systems of early days. A modern OS is very sophisticated and does much more than what we have

discussed above. In the next few sections, we will discuss some of the common concepts and terms related to the operating systems of modern computers.

## BATCH PROCESSING

Batch processing is one of the oldest methods of running programs that is still being employed by many data processing centres for processing their jobs. It is based on the idea of automatic job-to-job transition facility provided by almost all operating systems. In a batch mode, each user prepares his program off-line and submits it to the computer centre. A computer operator collects the programs which have been punched on cards and stacks one program or job on top of another. When a batch of programs have been collected, the operator loads this batch of programs into the computer at one time where they are executed one after another. Finally, the operator retrieves the printed outputs of all these jobs and returns them to the concerned users.

Batch processing is also known as serial, sequential, off line, or stacked job processing. When a computer is used in this way, the input data (and often the program) are introduced into the computer and processed automatically, generally without operator's intervention. Often many different jobs (or sets of data) are processed, one right after another, or even at the same time, but without any interaction from the users during program execution.

The method of batch processing reduces the idle time of a computer system because transition from one job to another does not require operator intervention. Moreover, it is the most appropriate method of processing for many types of applications such as payroll or preparation of customer statements where it is not necessary to update information (records) on daily basis. However, batch processing suffers from several disadvantages which are as follows :

1. It reduces timeliness in some cases. The time required to accumulate data into batches, in some instances, destroys much of the value of the data. The information that results from eventual processing is no longer timely.
2. Though efficient from the computer's point of view, batch processing makes each job wait in line at each step and often increases its turnaround time.
3. In batch processing, it is difficult to provide the desired priority scheduling. For example, if two high priority jobs were to be run but were in



separate batches, one would have to wait until the other's batch was completely processed.

## JOB CONTROL LANGUAGE (JCL)

We have just now seen that in batch processing, a set of jobs are stacked together and fed to the computer system. But the obvious question that arises to one's mind is that how the computer separates each job for automatic job-to-job transition. Moreover, how does the system know which compiler or what hardware devices are to be used by a particular job? In order that the operating system can identify a new job and determine what action should be taken for the job, some control information is necessary. These control statements are written in a language known as the *job control language* (JCL). Usually every program and data sets are preceded and followed by JCL statements.

When a program is prepared for a computer run, it is necessary to prepare job control statements and place them in proper order along with the program, before the program is fed to the computer system. Thus, each program has, besides the program itself, a set of instructions called JCL instructions which instruct the operating system on the identity and requirements of the job. JCL statements tell the OS such things as the name of the job, the user's name and account number, the I/O devices to be used during processing, the assembler or compiler to be used if language translation is required, and so on.

The job control language for one computer is different from that of another computer. Therefore, JCL statements differ from computer to computer and hence it is not possible to list here a set of statements that you can simply copy and use with your computer system. In order to know the JCL statements of your computer, it is suggested that you consult your instructor or supervisor, or an experienced programmer of your computer installation. However, a simple example is given below so that you can see what JCL statements look like. You will then have a better understanding of what the JCL statements used with your system do.

Suppose we wish to run a simple FORTRAN program on the Burroughs-6700 computer. The structure of the job deck in that case will be as follows:

```
<I> BEGIN JOB ONGC
```

```
<I> USER = SINHA/SNH
```

```
<I> COMPILE UPDATE FORTRAN GO
FORTRAN source program cards
```

```
<I> DATA
Data Cards
```

```
<I> END JOB
```

The above control statements are written in the work-flow language (WFL) which is the JCL for Burroughs-6700 system. Each control statement of WFL starts with the symbol <I> in column 1. The symbol <I> represents an invalid character (such as a combination of 1, 2 and 3 punched in the same column). This implies that no program or data card should have the symbol <I> punched on its first column. It should be noted that this symbol is typical and not universal. For example, IBM operating systems use two consecutive slashes to indicate a JCL statement and Honeywell-6000 uses a \$ character in the first column for this purpose.

The BEGIN JOB statement signals the beginning of a new job to the operating system. A name must be assigned to the job which is ONGC in this example. The next WFL statement is used to specify the usercode and the password of the user. Each user of a Burroughs-6700 installation is assigned a usercode by the system manager. In addition to the usercode, each user is also allotted a password which can be subsequently changed by the user as and when he desires. Both the usercode and his password must be specified correctly in the USER statement of WFL. In our example, the usercode is SINHA and the password is SNH. The password provides an added security to the user so that another person cannot run a job using his usercode. The operating system first checks the usercode given by the user with the list of valid usercodes stored within the system. If the usercode is found to be valid, then his password is checked to see if it is valid for the given user. Unless a user quotes both the usercode and his password correctly, the job will not be accepted by the operating system. The third WFL statement is a COMPILE statement. The word FORTRAN in this statement indicates that the source program is written in FORTRAN and hence a FORTRAN compiler will be required for its translation. The compiler specified by the user (in our case FORTRAN) is fetched from the secondary storage device (normally disk) and placed in the main memory to facilitate translation of the source program. The user also specifies the name of the object program which is UPDATE for our example. Hence the object code obtained after the successful compilation of the source program will be stored in the system by the name UPDATE. Finally, the word GO in this statement indicates that the object program is to be executed. The operating system will automatically start executing the program as soon as the translation process is over. It may be noted here that since we are compiling a program, so if any error is detected during the



translation process, no object code will be created for this program and subsequently no execution will be done even if the user has specified GO in the COMPILE statement of WFL. The COMPILE statement is followed by the source program statements. The DATA statement which is again a WFL statement denotes the end of the source program and the beginning of the data that will be used by the program. Finally, the END JOB statement indicates the end of the job. A job deck prepared in the above fashion can be submitted to a Burroughs-6700 installation for compilation and execution of a FORTRAN program.

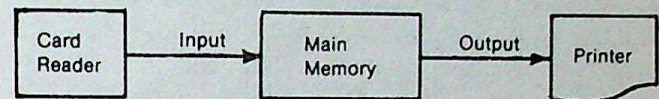
## SPOOLING

In batch mode of operation, the processing speed of a computer system can be further enhanced by a technique known as SPOOLing (Simultaneous Peripheral Output On Line).

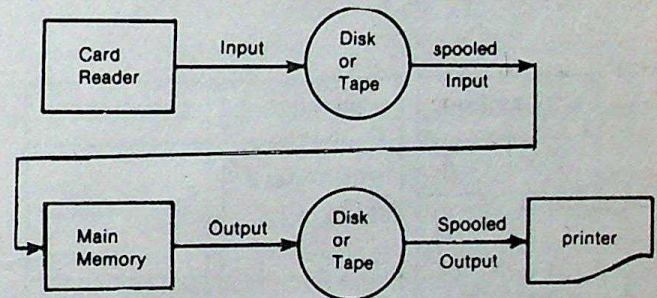
We know that dedicated I/O devices (devices which cannot be shared concurrently by several processes) like card readers and printers are considerably slower compared to the speed of the CPU. For example, a fast card reader can operate at a speed of 1500 cards/minute for an 80-column card. This is equivalent to reading 2000 bytes/second ( $1500 \times 80 \times 1/60$ ). On the other hand, for a medium speed computer, the processing speed of the CPU is approximately  $3 \times 10^6$  bytes/second. Thus while a card reader is supplying just one character to the main memory, the CPU can perform many thousands of internal operations before it needs to become involved with next character from the card reader. In other words, during reading of information into memory the CPU had to wait because of slow reading. The speed mismatch in this case is of the order of 1500. Similar speed mismatch exists for a large number of peripheral devices such as printers, teletypes, etc.

Spooling is a technique that has been successfully used on a number of computer systems to reduce the above mentioned speed mismatch and in turn the idle time of the CPU. It is the process of placing all data that comes from an input device or goes to an output device on either a magnetic tape or disk. This is shown in Figure 14.2. A batch of program when fed to the card reader is read and temporarily stored on a magnetic tape or disk instead of being directly stored in the main memory. The programs stored on tape or disk are now fed to and processed by the main computer. The results obtained are again written on tape or disk instead of being directly printed on the printer. The contents of output tape or disk are later printed on the printer. The process of storing the input data and output results on tape or disk is known as spooling. The primary

reason for spooling is to keep the programs and data readily available to the fast and expensive CPU on a high speed I/O medium such as tape or disk. Reading from tapes or disk is usually at the rate of  $10^5$  bytes/second in contrast to 2000 bytes/second for a card reader. Similarly, the speed of writing on tape or disk is of the order of  $10^5$  bytes/second contrasted to writing on even a fast line printer at a speed of 2000 bytes/second.



(a) Mode of data transfer without spooling facility.



(b) Mode of data transfer with spooling facility.

Figure 14.2. Illustrating the process of spooling.

Special spooling programs are executed by the operating system to transfer the data from the disk or tape to the main memory or an input or output device. In a sense, the disk or tape device acts as a buffer area between main storage, which is extremely fast, and I/O devices, which are relatively slow. Spooling programs are executed when the CPU is not too busy with other jobs. However, in most computer systems, special low cost I/O processors are used for spooling the input data from a card reader (or any other slow input device) on to the tape or disk or for printing the spooled results from the tape or disk on to the printer (or any other slow output device). These I/O processors function independent of the main CPU. This enables the main high speed expensive CPU to be fully devoted to main computing job.



The process of spooling is transparent to the user's program. In general, spooling makes better use of both the main memory and the CPU.

## MULTIPROGRAMMING

In case of batch processing, the batched programs are loaded one after another in sequence into the main memory for processing. Once loaded, a program will remain in the main memory until its execution is completed. Thus, the program which is currently being executed will be the sole occupant of the users' area of the main memory (remember that the supervisor always resides in a part of the main memory) and it will have the CPU exclusively available to itself. When there is only one program in main memory, two of the system's most powerful resources may be under-utilized, its expensive memory and the full capabilities of the CPU. As shown in Figure 14.3, every program will not be large enough to occupy the full users' area of the main memory. Similarly, all programs will not be highly computational to utilize the full processing capability of the CPU. Basically there are two types of programs :

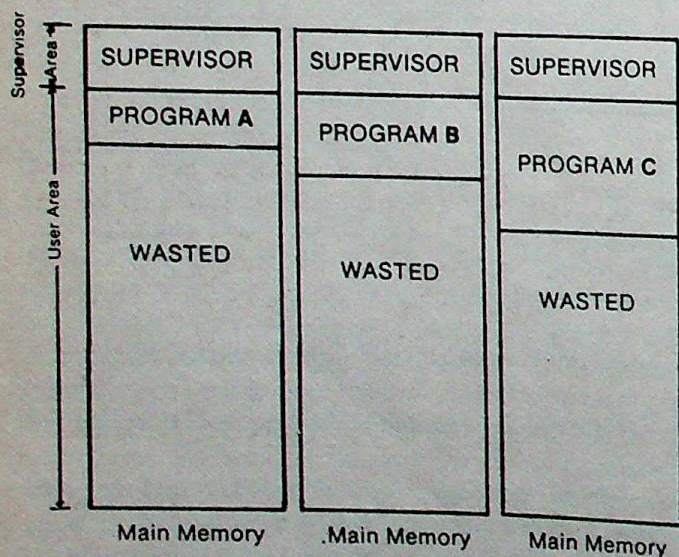


Figure 14.3. Illustrating the under-utilization of memory when only one program occupies the whole memory.

I/O-bound programs and CPU-bound programs. Programs used for commercial data processing normally read in vast amount of data, perform very little computation and output large amount of information. Such programs are known as *I/O-bound programs*, since the majority of work they perform is input-output. On the other hand, programs used

for scientific and engineering applications need very little I/O but require enormous computation. These programs are called *CPU-bound programs* because more of CPU-time is required for processing such programs.

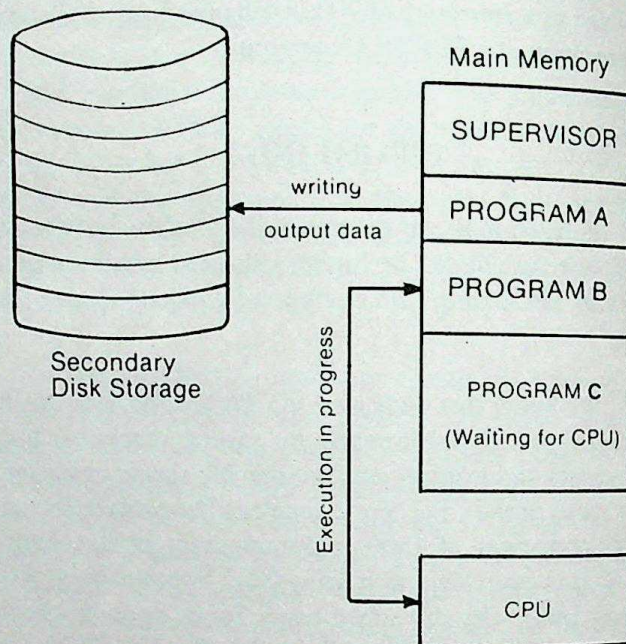
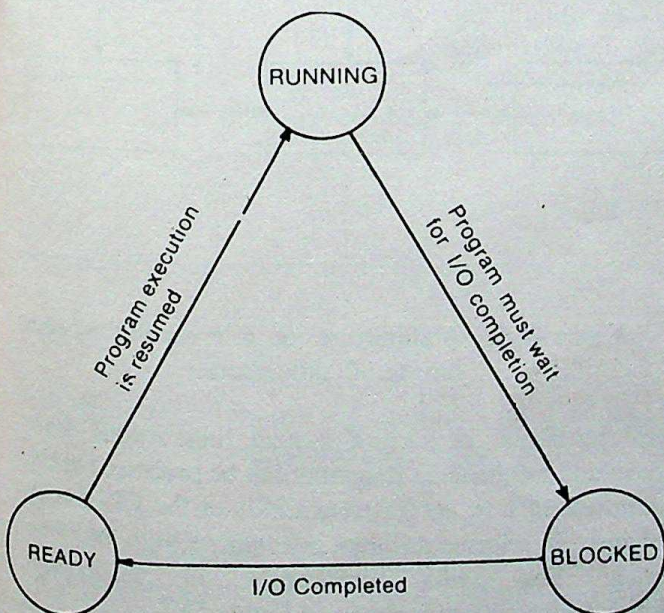


Figure 14.4. illustrating the operation of multiprogramming.

In order to overcome the problem of under-utilization of main memory and the CPU, the concept of multiprogramming was introduced in operating systems. "Multiprogramming is the name given to the interleaved execution of two or more different and independent programs by the same computer." With the storage-resident supervisor concept (see Figure 14.3) we have been introduced to the notion of having two programs in the main memory at the same time : the supervisor for overall system control and the user program for performing user's task. In multiprogramming, this concept is carried one step further by placing two or more user's programs in main memory and executing them concurrently. The CPU switches from one program to another almost instantaneously. Since the operating speed of CPU is much faster than that of I/O operations, the CPU can allocate time to several programs instead of remaining idle when one is busy with I/O operations. In multiprogramming system,



when one program is waiting for I/O transfer, there is another program ready to utilize the CPU, thus it is possible for several users to share the time of the CPU. However, it is important to note that multiprogramming is not defined to be the execution of instructions from several programs at the same instant of time. Rather, it does mean that there are a number of programs available to the CPU (stored in main memory) and that a portion of one is executed, then a segment of another and so on. Although two or more users' programs reside in the main storage simultaneously, the CPU is capable of executing only one instruction at a time. Hence at any given time, only one of the programs has control of the CPU and is executing instructions. Simultaneous execution of more than one program with a single CPU is impossible. In some multiprogramming systems, only a fixed number of jobs can be processed concurrently (multiprogramming with fixed tasks) (MFT), while in others the number of jobs can vary (multiprogramming with variable tasks) (MVT).



A simple example of multiprogramming is given in Figure 14.4. At the particular time instance shown in the figure, program A is not utilizing the CPU since it is busy writing output data on to the disk (I/O operation). Hence the CPU is being utilized to execute program B which is also present in the main memory. Another program C, residing in the main memory, is waiting for the CPU to become free. Actually, as shown in Figure 14.5, in case of multiprogramming all the programs residing in the main memory will be in one of the following three states : running (CPU is being used), blocked (I/O operation is being done) and ready (waiting for CPU). For our example, programs A, B and C are in blocked, running and ready states respectively. Since program C is in the ready state, as soon as the execution of program B is completed or program B requires to do I/O operation, the CPU will start executing program C. In the meanwhile, if A completes its output operation, it will be in the ready state waiting for the CPU. Thus in multiprogramming, the CPU is almost always busy. When program A is reading data or outputting results (I/O operations), program B's instructions can then be executed, and if both programs are involved in I/O activity, then program C can be executed. The area occupied by each program residing simultaneously in the main memory is known as a *memory partition*. The actual number of memory partitions and hence programs allowed in the main memory at any given time varies depending upon the operating system in use at a particular installation. Moreover, those jobs awaiting entry into the main memory are queued on a fast secondary storage device such as a magnetic disk. The first job from this queue will be loaded into the main memory as soon as any one of the jobs already occupying the main memory is completed and the corresponding memory partition becomes free.

### Requirements of Multiprogramming Systems

Multiprogramming has two main advantages: increased throughput and lowered response time. Throughput is increased by utilizing the idle time of the CPU for running other programs that are already residing in the main memory. Response time is lowered by recognizing the priority of a job as it enters the system and by processing jobs on a priority basis.

On the other hand, the incorporation of multiprogramming in the operating system has, of course, complicated matters. For a computer to work simultaneously on many programs, the following additional hardware and software features are required :

1. **Large memory.** For multiprogramming to work satisfactorily, large main memory is required (of

Figure 14.5. The three different states of a program residing in main memory in case of multiprogramming.



the order of 128K or more), together with fast secondary storage devices like disk and fast CPU. The main memory should be large enough to accommodate a good number of users' programs along with the supervisor.

2. **Memory protection.** Computers designed for multiprogramming must provide some type of memory protection mechanism to prevent a program in one memory partition from changing information or instruction of a program in another memory partition. For example, in Figure 14.4 we would not want program A to inadvertently destroy something in the completely independent program B or program C. In a multiprogramming environment this is achieved by the *memory protection feature*, a combination of hardware and software, which prevents one program from addressing beyond the limits of its own allocated storage area.
3. **Program status preservation.** We have seen that in multiprogramming, a portion of one program is executed, then a segment of another, and so on. This requires the stopping of a program execution and then restarting its execution after some time. In order to restart a program, all the values that were stored in memory and the CPU registers that were being used at the time of its stopping should be restored. A new program would however need all CPU registers for its use and would clear them. Thus, before a program is suspended and the control is passed to another program, the values of all CPU registers (pc, accumulator, etc.) should be stored in the memory area of that program and then restored when the control is ultimately returned to the first program. This is known as program status preservation.
4. **Proper job mix.** A proper mix of I/O-bound and CPU-bound jobs is required to effectively overlap the operations of the CPU and I/O devices. It is necessary that when a program is waiting for I/O operation, another program must have enough computation to keep the CPU busy. If all programs need I/O at the same time, the CPU will again be idle. Hence the main memory should contain some CPU-bound programs and some I/O-bound programs in its various partitions so that at least one of the program which does not need I/O is always available to the CPU for processing.

## MULTIPROCESSING

Upto this point we have considered systems with a single CPU. However, we have already seen that the use of I/O channels or I/O processors improves the efficiency of the computer system by making possible concurrent input, processing, and output operations. The CPU can perform arithmetic and logical operations on parts of one or more programs while I/O operations are concurrently carried out by I/O processors on other parts of programs. The architecture of a computer system having I/O processors is shown in Figure 14.6.

The idea of use of I/O processors to improve the performance of a computer system was carried one step further by designing systems that make use of more than one CPU. Such systems are called multiprocessing systems. The term *multiprocessing* is used to describe interconnected computer configurations or computers with two or more independent CPUs that have the ability to simultaneously execute

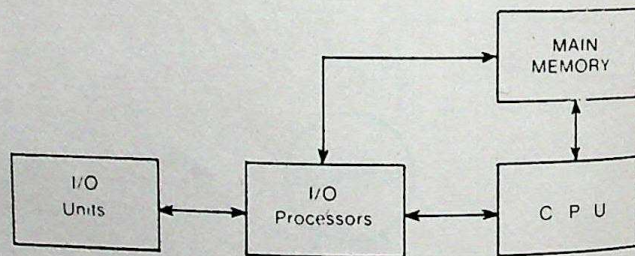


Figure 14.6. Architecture of a computer system having I/O processors.

several programs. In such a system, instructions from different and independent programs can be processed at the same instant of time by different CPUs or the CPUs may simultaneously execute different instructions from the same program. The basic organisation of a typical multiprocessing system is shown in Figure 14.7.

There are almost limitless number of possible multiprocessing systems. In some systems, several small CPUs are linked together to perform the major processing. If one of the small CPUs breaks down, the other CPUs will automatically take over its job. In other systems, CPUs are connected into elaborate computer networks. Distributed data processing (discussed in next chapter) is an example. In these networks, small CPUs called front-end processors



are used for scheduling and controlling all jobs entering the system from remote terminals and other input devices. Thus, the main CPU or CPUs called host computers or back-end processors are used only for major processing jobs and not for data communication. In some multiprocessing systems, each CPU performs only specific

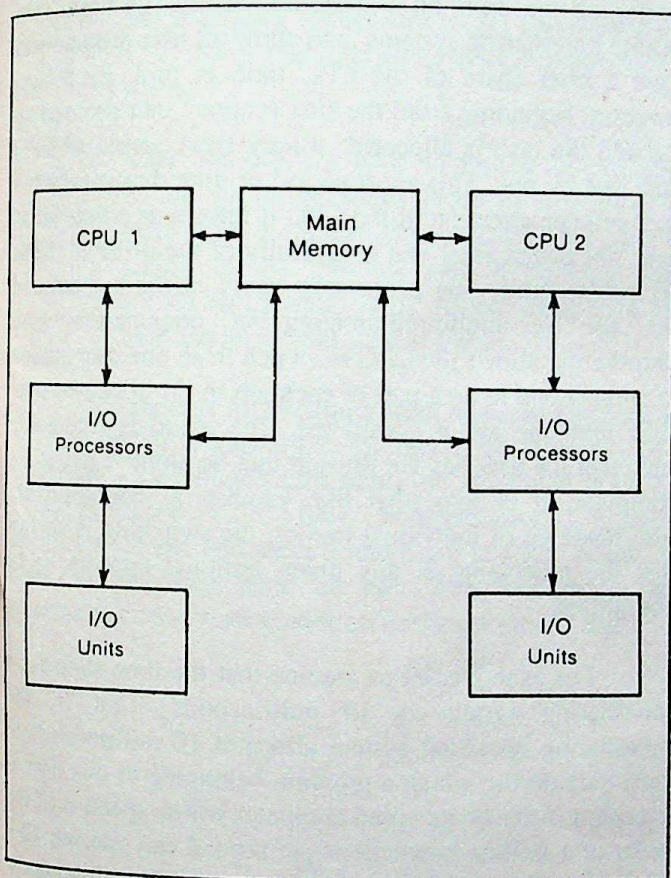


Figure 14.7. Basic organisation of a typical multiprocessing system.

types of applications. For example, in case of a multiprocessing system with two CPUs, one may be used to process only on-line jobs while another one may be meant for processing only batch applications. However, these systems are so designed that in case of breakdown of one CPU, the other CPU takes over the complete workload until repairs are made. Moreover, different multiprocessing systems use different types of memory configurations. In some systems each CPU has its own main memory, in others all the CPUs may share a common memory, while in some others each CPU may have access to both separate and common memories.

## Difference Between Multiprogramming and Multiprocessing

Multiprogramming is the interleaved execution of two or more processes by a single CPU computer system. On the other hand, multiprocessing is the simultaneous execution of two or more processes by a computer system having more than one CPU. To be more specific, we may point out here that multiprogramming involves executing a portion of one program, then a segment of another, etc., in brief consecutive time periods. Multiprocess design, however, makes it possible for the system to simultaneously work on several program segments of one or more programs.

## Advantages and Limitations of Multiprocessing

There are numerous advantages of multiprocessing some of which are listed here :

1. It improves the performance of computer systems by allowing parallel processing of segments of programs. Better performance is directly reflected by increased throughput and lowered turnaround time of such systems.
2. In addition to the CPUs, it also facilitates more efficient utilization of all the other devices of the computer system.
3. It provides a built-in backup. If one of the CPUs breaks down, the other CPU(s) automatically takes over the complete workload until repairs are made. Thus, a complete breakdown of such system is very-very rare.

Multiprocessing, however, is not an easy task because of the following reasons :

1. A very sophisticated operating system is required to schedule, balance and coordinate the input, output and processing activities of multiple CPUs. The design of such an OS is a time taking job and requires highly skilled computer professionals.
2. A large main memory is required for accommodating the sophisticated operating system along with several users programs.
3. Such systems are very expensive. In addition to the high charges paid initially, the regular operation and maintenance of these systems is also a costly affair.



It is expected that multiprocessing systems will soon become commonplace. In the future, all large computer systems will use multiple, parallel processors to share high-speed and complex operations and to enhance processing throughout. There will be computers within computers, because some of these processors will be complete microcomputers. The entire system will be under the control of a complex, powerful operating system.

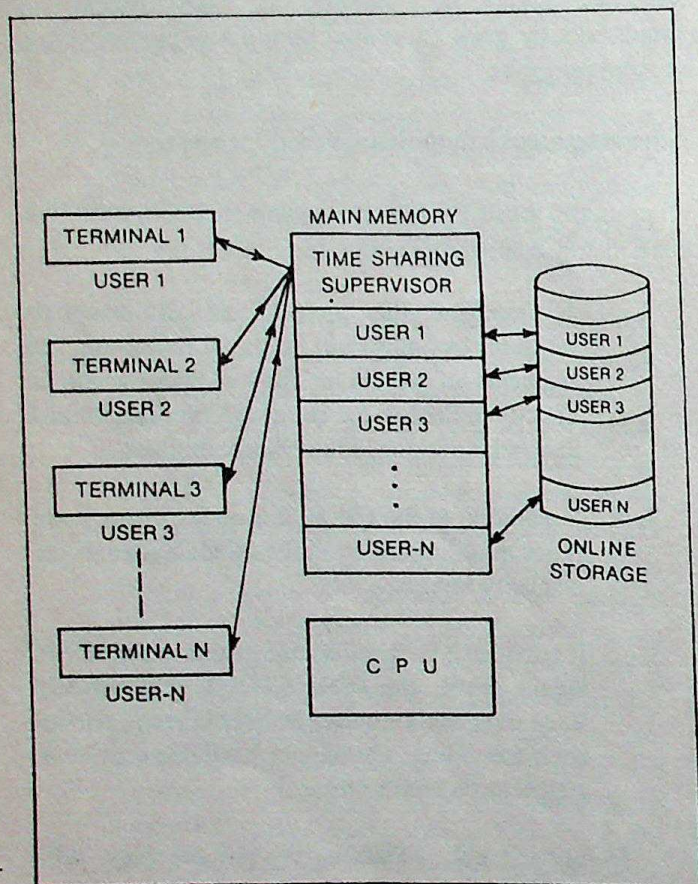


Figure 14.8. Concept of a timesharing system.

## TIME-SHARING

*Timesharing* is a term used to describe a processing system with a number of independent, relatively low speed, online, simultaneously usable stations. Each station provides direct access to the CPU.

In other words, timesharing refers to the allocation of computer resources in a time-dependent fashion to several programs simultaneously. The principal notion of a

timesharing system is to provide a large number of users direct access to the computer for problem solving. This is accomplished by providing a separate terminal to each user. All these terminals are connected to the main computer system. Thus, a timesharing system has many, even hundreds, of terminals linked up to the same computer at the same time. This is shown in Figure 14.8. Unlike multiprogramming, where programs are executed on a priority basis, in timesharing the CPU time is divided among all the users on a scheduled basis. The basic idea behind timesharing systems is to allow all user programs to have a brief share of the CPU time in turn. Each user program, beginning from the first program and proceeding through the last, is allocated a very short period of CPU time one by one. This short period of time during which a user gets the attention of the CPU is known as a *time slice*, *time slot* or *quantum* and is typically of the order of 10 to 20 milliseconds. The processing speed of the system and the use of multiprogramming in conjunction with timesharing allows the CPU to switch from one user station to another and to do a part of each job in the allocated time slice until the job is completed. The speed is frequently such that the user has the illusion that he alone is using the computer. It is somewhat like viewing a motion-picture film made up of individual frames, the switching is so fast that the processing at any given terminal appears to be continuous.

For example, let us assume that the time slice for a timesharing system is 10 milliseconds. That is, the timesharing operating system allocates 10 milliseconds to each user during which a program belonging to this user is executed. An average speed computer whose speed is of the order of 1 million instructions per second can execute  $10 \times 10^{-3} \times 10^6 = 10,000$  instructions in 10 milliseconds. Suppose there are 100 users for this timesharing system. Then if 10 milliseconds is allocated to each user, a particular user will get the CPU's attention once in every  $10 \times 100$  milliseconds = 1 second. As human reaction times are a few seconds, a particular user will not notice any delay in executing his commands and normally feels that he is the sole user of the system. Moreover, it is not economically feasible to allow a single user to use a large computer interactively because his speed of thinking and typing is much slower than the processing speed of a computer. While a particular user is engaged in thinking or is busy typing his input, a timesharing system can service many other users.

Even though it may appear that several users are using the computer system at the same time, a single CPU system can only execute one instruction at a time. Thus like a multiprogramming system, even with a timesharing system, only one program can be in control of the CPU at



any given time. As a result, at any instant, all the users who are using a timesharing system will fall in one of the following three status groups.

1. **Active** : the user's program currently has control of the CPU. Obviously only one user will be active at a time.
2. **Ready** : the user's program is ready to continue but is waiting for its turn to get the attention of CPU. More than one user can be in ready state at a time.
3. **Wait** : the user has made no request for execution of his job or the user's program is waiting for some I/O operation (for instance, the user is sitting at the terminal and is thinking what should be the next step). Again, more than one user can be in wait state at a time.

The process of switching from one status to another is illustrated in Figure 14.9. Moreover, the concepts of a timesharing system and user status are illustrated in Figure 14.10. In Figure 14.10(a) user 2 is active, users 1, 3, and 4 are in wait status and users 5 and 6 are in ready status. As soon as the time slice of user 2 is completed, the timesharing supervisor moves on to the next ready user

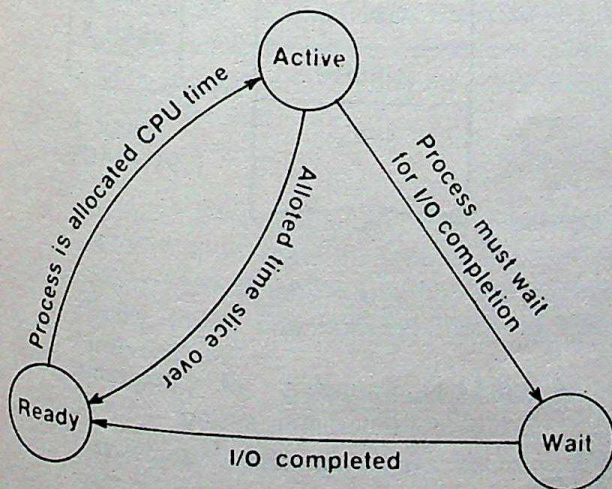


Figure 14.9. The process of switching between the three status of a time sharing system.

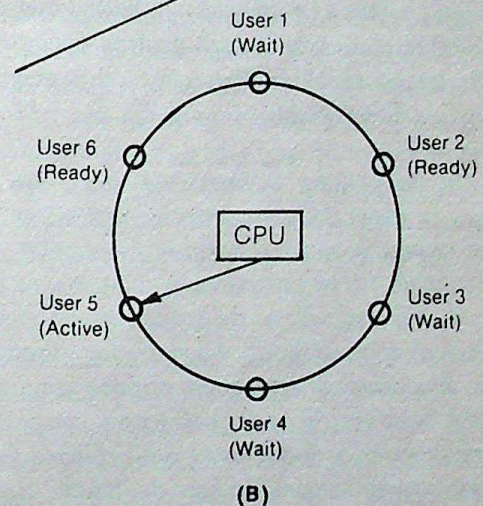
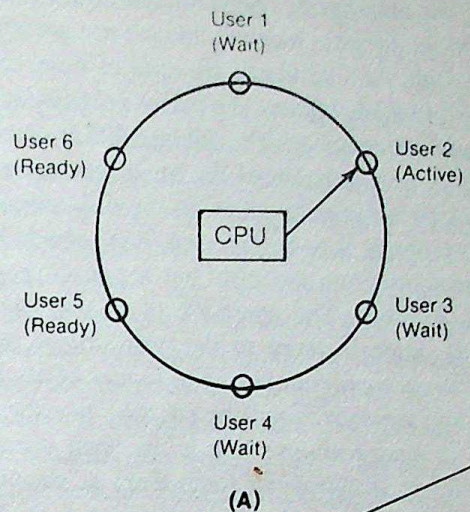


Figure 14.10. User status in a timesharing system.

(those in wait status are skipped since they are making no demand for the CPU). The next ready user in the queue is user 5 which now becomes active as shown in Figure 14.10(b). User 5 will remain active until the allotted time slice expires, or until the program needs I/O operation, or if the program execution is over during this time period. At that time, control is passed on to the next ready user in the queue which is user 6 for our example. Whenever I/O operation is completed for a wait user, that user's status will be changed to ready and serviced the next time around.

In a typical timesharing system, hundreds of users may be using the system simultaneously. As the total main memory available in a computer is limited, it is not possible to keep the programs of all the users of a timesharing system simultaneously in the main memory. Thus at any



instant, the timesharing operating system keeps only a few programs in the main memory and the rest are stored on the disk storage. At a particular instance of time, the memory resident programs include the active program and some of the ready programs which will get CPU's attention very shortly. A wait program of the main memory is normally replaced by a ready program on the disk storage. As and when a program is to be executed, it is brought back to the main memory from the disk and the inactive program is sent to the disk. The operation of transferring programs from the main memory to the disk storage and back is known as *swapping*. Referring to Figure 14.10, as user 2 is executing, the system will be ensuring that the next ready job is in main memory. If it is not, then one of the wait programs is swapped out (onto disk) as shown in Figure 14.11(a) and the next ready user, user 5 in this case, is swapped in as shown in Figure 14.11(b). This swapping process, sometimes known as *roll-in roll-out*, is repeated many times within a few seconds. In this case, disks are the only feasible secondary storage devices since they have a much faster rate of information transfer than magnetic tapes and provide direct access.

In timesharing systems, the user often carries a dialogue or conversation with the central system. Hence it is also known as *conversational* or *interactive* computing. The computer can be programmed to interrogate the user as required, to respond to requests, replies, and even to mistakes. A user can proceed step-by-step, testing portions of his procedure or trying out various approaches to a problem solution. This is the reason why timesharing systems have been found to be most suitable for program development and testing. In fact, the BASIC language was designed specifically for timesharing systems. Systems which are fully interactive inspect each statement of a new program as it is entered into the computer via a terminal. Any syntax errors in the usage of the language are detected and immediately displayed on the video screen so that the user can make appropriate corrections. The user corrects his program with an editing system. When all the syntax errors are corrected, the program can be run and tested to ensure its validity. The requisite data is fed from the terminal during program execution. The data may also be stored in a data file on disk and fed to the program when it needs it. Errors encountered during execution of the program are displayed on the terminal. These run time errors can be immediately corrected by the user and another test run can be made. The greatest benefit of such a system is that errors can be encountered, corrected, and work can continue immediately. This is in contrast to a batch system in which errors are corrected offline and the job is submitted for another run. The time delay between job submission and return of the output in a batch system is often measured in hours. Overnight turnaround is also very common.

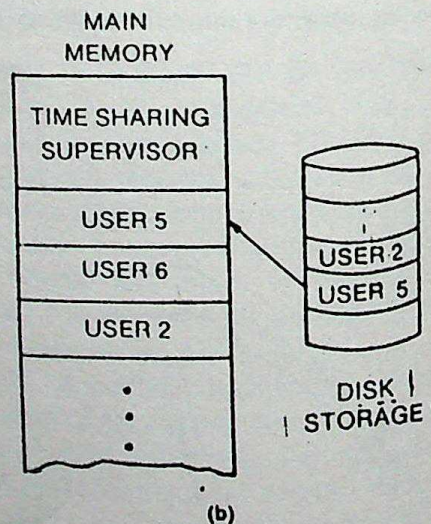
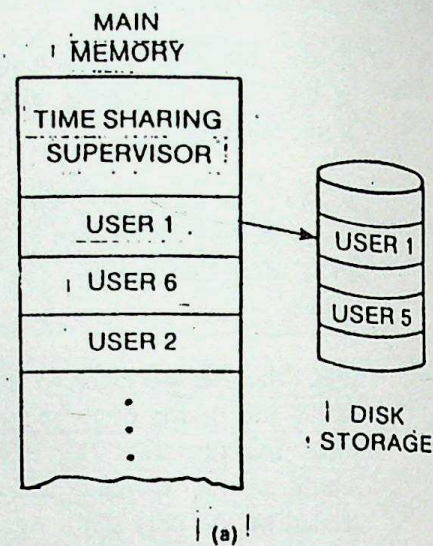


Figure 14.11. Swapping of programs in a timesharing system.

Obviously this time lapse does not contribute well to the thinking efficiency of a programmer. The interactive programming and debugging capability of BASIC has proven to be so effective in improving programmer efficiency that interactive versions of the batch-oriented FORTRAN and COBOL are also available now.



### Advantages of Timesharing

1. *Reduces CPU idle time.* It is wasteful and expensive for the CPU to be effectively utilized less than 30% of the time. Yet this is what happens in a conventional batch processing installation as the CPU waits during set-up times and during I/O operations. Timesharing significantly increases CPU utilization by switching from one program to another in rapid succession. Thus the throughput of the installation increases to a great extent.
2. *Offers computing facility to small users.* Small users can gain direct access to much more sophisticated hardware and software than they could otherwise justify or afford. In a timesharing system they merely pay a fee for resources used and are relieved of the hardware, software, and personnel problems associated with acquiring and maintaining their own installation.
3. *Provides advantages of quick response.* The turnaround time or the response time is negligible in case of a timesharing system. Thus, timesharing allows managers to react more rapidly. Furthermore, it permits them to interact or converse with the system in seeking solutions to unusual problems and answers to poorly defined questions. Timesharing may also reduce waste in the use of business resources and it can permit quick follow-up on creative ideas. In short, it helps in improving the users' efficiency to a great extent.
4. *Reduces the output of paper.* If a manager can retrieve at any time the specific information he needs from an online file, he does not need a bulky report that contains much of the file information.
5. *Avoids duplication of software.* There are several programs which are frequently used by many users. In a time-sharing system, such programs are stored in the system library. A user need not write his own program instructions when performing such processing tasks. He need only call up the needed program stored online at the computer site and supply the data.

### Disadvantages of Timesharing

1. *Question of security.* Since hundreds of users use a timesharing system simultaneously, provision must be made to protect the security and integrity of user programs and data. The programs and data of different users should not get mixed up. This is currently being accomplished by such methods as (a) assigning and (b) requiring hierarchies of passwords or lockwords from users prior to file access. However, in spite of such precautions, skilled penetrators succeed in bypassing the programmed controls of current timesharing systems.
2. *Problem of reliability.* A time sharing system should be highly reliable as it caters to the needs of several users. Hence provisions must be made to provide dependable and continuous service. The self-repairing computer or some sort of standby arrangement may ultimately help to overcome the reliability problem. But troubles that occur are often with the online peripheral devices or software and not with the main CPU.
3. *Problem of data communications.* In a timesharing system, the users interact with the main computer system through remote terminals that require data communication facilities. The cost of data communication has been declining but not so rapidly as the cost of data processing. Thus, data transmission charges make up an increasing portion of the total timesharing cost package. In addition, telephone lines were designed for voice communication rather than data communication, with the result that current transmission facilities are not considered adequate by many timesharing spokesmen.
4. *Question of overhead involved.* The reader can probably appreciate that the timesharing system with its control functions such as switching from user to user and swapping programs in and out takes up an appreciable amount of CPU time. This is termed overhead and must be minimized in the overall hardware-software design. If a system is properly balanced, then the overhead is manageable and the computer response time to a user request will be small. However, if the system is overloaded with too many users, then the overhead can get out of hand, resulting in very poor response.



## ON-LINE PROCESSING

On-line processing (also called direct-access or random-access processing) permits transaction data to be fed under CPU control directly into secondary on-line storage devices from the point where data originates without first being sorted (sorting of data is required in case of sequential processing). These data may be keyed in by the use of a typewriter like terminal, or they may be produced by a variety of other data collection and transaction recording devices. The CPU can make programmed input control checks during this process. Using these input data, appropriate records (which are normally organized in the secondary storage unit in random fashion) may be quickly updated. The access to, and retrieval of, any record is quick and direct. Information contained in any record is accessible to the user without the necessity of a sequential search of the file and within a fraction of a second after the enquiry message has been transmitted. Thus, on-line processing systems feature random and rapid input of transactions and immediate and direct access to record contents as and when needed. A simplified concept of on-line processing is depicted in Figure 14.12.

A timesharing system is a typical example of on-line processing. However, it should be noted here that on-line processing systems may differ considerably in level of complexity. Some systems may have only a few terminals, and the volume of transactions to be processed may thus be low; these transactions may be processed on a first-come, first-served basis with no attempt being made to use timesharing or multiprogramming and the system may employ relatively simple data communication facilities. At

On-line processing and direct access to records require unique hardware and software. For example, the capacity of the primary storage unit of the system must be adequate to accommodate the complex on-line operating system supervisor along with other users' programs. Also, since many on-line users may have access to stored records, software security provisions are necessary to prevent confidential information from falling into unauthorized hands and prevent deliberate or accidental tampering of data and program files. Furthermore, in many cases, CPU must be fast enough to respond to multiple on-line stations operating simultaneously in a multiprogramming mode; and large capacity peripheral on-line storage units are required to store additional operating system elements, user data and programs. Finally, data transmission facilities must be provided to communicate with on-line terminals located in the next room, on the next block, or thousands of miles away.

## REAL-TIME PROCESSING

There are many applications that require an immediate response from the computer. Getting a stock market quotation, finding the current level of product inventory, and searching a criminal data file for a possible suspect may all be actions that need to be done without delay. In these cases, a real-time processing system is needed. Real-time means immediate response from the computer. A system in which a transaction accesses and updates a file quickly enough to affect the original decision making is called a *real-time system*. The essential feature is that the input data must be processed quickly enough so that further action can then be promptly taken on the results.

In other words, a real-time processing system may be described as an on-line processing system with severe time limitations. It may be noted here that a real-time system uses on-line processing, but an on-line system need not necessarily operate in real-time mode.

Real-time processing requires immediate (not periodic) transaction input from all input-originating terminals. Many remote stations are tied directly by high-speed communications equipment into one or more CPUs. Several stations may be operating simultaneously. Files are updated each minute, and enquiries are answered by split-second access to up-to-the-minute records. The system processes input data and presents the result in such a form that human judgement can immediately be brought into decisive action.

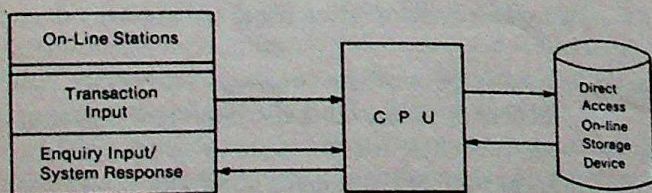


Figure 14.12. On-line processing.

the other extreme are on-line systems that have hundreds of remote stations and communication lines; they use multiprogramming or timesharing to keep the response time within acceptable range so that users do not get irritated due to delays in response.

One of the early and very sophisticated commercial real-time systems was the American Airlines SABRE



reservation system. The following factors justify the use of real-time processing for an airline reservation system :

1. There are hundreds of flights daily.
2. Each flight may have as many as 300 seats or more "in inventory".
3. As soon as a seat is reserved/cancelled, the concerned files must be updated before the next transaction can be processed.
4. The response time should be very short because a customer's reservation is to be done while he waits.
5. Seats may be sold for only a portion of flight. For example, Mr. XYZ may book a seat to Baroda on a Delhi to Bombay flight which stops in Baroda. That seat will then be available for the Baroda to Bombay leg.
6. Hundreds of agents throughout the country are selling seats from the inventory.
7. An airline seat is a very perishable item. If it is not sold, it is lost once a flight is made.

Owing to the above mentioned reasons, efficient operation of present-day major airlines would be very difficult and almost impossible without a real-time processing system. Few more examples of business real-time processing are :

1. Air traffic control system.
2. Reservation systems used by hotels and car rental agencies. These systems keep track of the availability of hotel rooms or cars at any instance of time.
3. Systems that provide immediate updating of customer accounts in saving banks.
4. Systems that provide up-to-the-minute information on stock prices.
5. Process control systems as in nuclear reactor plants and steel mills.

Similarly, there are many applications that require real-time processing. It would be a mistake, however, to assume that real-time processing should be universally applied to all data processing applications. A quick-

response system can be designed to fit the needs of the business. Some applications can be processed on a lower priority or background basis using batch methods (e.g. payroll); some can be on-line with periodic (not immediate) updating real-time methods.

Real-time systems are required to be highly reliable because even minimal downtime in many critical applications may be hazardous causing danger to several lives and substantial financial loss. For example, in case of a computerised air traffic control system, the radar and computers that keep track of air traffic in each region, must operate constantly with minimum maintenance. An unscheduled downtime in this real-time system may cost lives of several human beings. Similarly, in case of a chemical process-control system, high degree of reliability is essential, since even minimal downtime can lead to a ruined batch of product or damage to expensive equipment with substantial financial loss. In order to achieve the desired degree of reliability, real-time systems are normally duplicated so that, in the event of a break down, back-up facilities are immediately available for continuous operation of the system. This makes some systems very expensive but, in the environment to which real-time systems are applied, a fail-proof system is essential.

## VIRTUAL STORAGE

*Virtual storage* may be described as a hierarchy of two memory systems - one of them is a low cost, large capacity, low speed system (on-line disk storage) and the other is a high cost, small capacity, high speed system (main memory). The operating system manages the two memory systems in such a way that a user feels that he has access to a single, large, directly addressable, and fast, main memory.

A virtual memory system facilitates its users to use a large addressable memory space without worrying about the size limitations of the physical main memory. Moreover, in case of multiprogramming or timesharing systems, it also permits the sharing of memory space among several users efficiently and economically.

In order to implement a virtual memory system, the main memory is divided into fixed size contiguous areas, called *page frames*. In addition, all users programs, residing on the on-line disk storage, are also divided into pieces of the same size, called either *pages* or *segments*. Now, only those program pages or segments that are actually required at a particular time in the processing, need be in the primary (or real) storage. The remaining pages or segments may be kept temporarily in *online (or virtual) storage*, from where they can be rapidly retrieved as and when needed following



program interruption (see Figure 14.13). The operating system handles the swapping of program pages or segments between the main memory and the on-line disk storage.

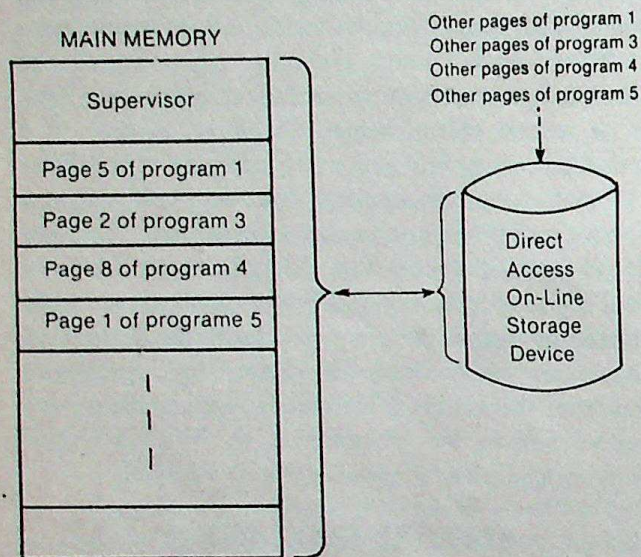


Figure 14.13. A virtual storage system.

Thus, from the applications programmer's point of view, the effective (or virtual) size of the available primary storage may appear to be unlimited.

In previous sections, we have seen that when a computer system is operated in timesharing or real-time mode, only a few instructions are executed for one user at any given moment. Then the computer executes a few instructions for another user and so on. In a few seconds, the computer could process a few instructions for over a hundred users. Because the computer is only executing a few instructions of one user's program at one time, it will be wastage of main memory if the complete program of that user is stored in main memory, instead, the concept of virtual storage can be effectively used in this case to simultaneously accommodate program segments of a large number of users in the main memory. With more program segments of different users residing simultaneously in main memory, the CPU is less likely to have to wait for programs to be transferred from the disk to main storage. This reduces CPU idle time and increases the number of jobs that can be run in a given time span. It is important to note that even in this particular application of virtual storage, the

physical size of main memory remains the same. It only appears to be larger because more gets done in less time.

## OS-CONTROLLED SOFTWARE

A computer can do nothing with a program of instructions, and each job required must have its own special program. However there are many tasks of a routine nature that all computer users require their machine to perform from time to time. It would clearly be wasteful if each user spent a lot of time writing programs for these tasks and it is normal practice for the computer manufacturers to supply programs for these tasks along with the operating system and the hardware of the machine. These OS-controlled *softwares* reduce the time and expense of preparing applications programs and are normally grouped in three categories - translating programs, library programs, and utility programs. They are briefly discussed below.

### TRANSLATING PROGRAMS

Translating programs, also known as language processors, are system programs that translate a source program written by the user to an object program which is meaningful to the hardware of the computer. These include the assembler and the various compilers and interpreters available with the system. A translating program is usually called up from a direct-access storage device only after the job control program of the operating system interprets a job control statement and informs the operating system supervisor of what is needed.

### LIBRARY PROGRAMS

Library programs consist of frequently used subroutines supplied by users and computer vendors. These standard routines are stored in a direct-access storage device and are called up by the operating system whenever they are required in the processing of other programs. This eliminates the need for a programmer to rewrite these modules every time they are used. A *librarian program* controls the storage and use of these programs in the system library. It maintains a program directory for this purpose and also facilitates the addition of new programs to the library or deletion of unwanted programs from the library. In the area of scientific applications, the usual types of library routines available are the mathematical functions such as square root and exponential functions. Other operations of various types are also encountered such as matrix inversion, statistical analysis, conversion of numbers from one base to another (binary to decimal and decimal to binary), etc. All these and many other frequently used routines are normally available as library programs.



## UTILITY PROGRAMS

Utility programs, also known as service programs, are routines that perform needed services such as editing texts or programs, debugging programs to correct logical mistakes, sorting records into a particular sequence for processing, or transferring data from one I/O device to another. These routines are also available for call-up by the operating system and once again the job control statements of a particular user tell the operating system supervisor which utility programs are needed by the user. A few examples of utility programs commonly available in a computer system are briefly described below.

1. *Text editor.* A text editor is a program that facilitates the creation and correction of texts. The text being edited could be an English language letter, but most often, it is a symbolic language program typed by the user. The text editor program does not interpret the meaning of the text but has the capability of changing it when special commands are issued by user. For example, when a symbolic language program is being entered into a computer memory via a video-terminal, the programmer may use the facility of a text editor program to correct his typing errors by issuing commands to insert, delete, or replace characters in his source programs. Thus, with the help of a text editor, the user can prepare programs and correct them with relative ease.

2. *Debugging tool.* Debugging tool is a program that helps the user to locate and correct logical mistakes in his program. A dynamic debugging tool allows the programmer to control program execution using a video terminal while his program is being executed in the computer. While his program is being executed, the user can stop the execution of the program at any desired point, he can examine contents of various registers, change contents of registers and memory, make alterations to his binary programs, and other similar functions. Thus, by using the facilities of a dynamic debugging tool, the user can easily detect and correct logical errors in his program.

3. *Sort and merge.* In the area of commercial data processing applications, the most widely used routines are sort and merge. Sort programs are used to arrange data into a specified sequence. For example, business transactions may be stored in computer in the order in which they occur. The transactions may have to be sorted by different items such as by account number to identify the customer or by salesman's name to calculate the commission to be paid. The sort program reads the unsequenced input file and by means of various copying techniques ultimately produces as output a copy of the input file in the required sequence. Merge programs, on the other hand, are used to combine

two or more sets of sorted data into one file containing all the items of all the original sets in sorted order.

4. *Memory dump program.* A memory dump program allows the user to print the contents of specified locations in main memory at some particular point during the program execution. A memory dump typically shows both the program and operand data. By inspecting both program and data, and comparing it with what it should have been if the program had run correctly, the programmer is able to find the mistakes in his program.

5. *Trace routine.* A trace routine allows the user to trace the flow of his program while it is executed. He can request, for example, that the contents of certain registers or memory locations be printed every time a branch statement is executed or when the value of certain variables are changed. This allows the user to get a clear picture of what his program is doing and thus be able to correct mistakes in his program.

6. *Peripheral interchange programs.* These utility programs facilitate transfer of data from one I/O device to another. They make possible the copying of data from one unit, for instance, magnetic tape, to another unit, for instance, magnetic disk. It is also possible to copy data from one tape unit to another tape unit or from one disk unit to another disk unit. This results in a more efficient utilization of the data preparation equipments.

## QUESTIONS

1. What is an operating system ? Why is it necessary for a computer system ?
2. List out the various functions normally performed by an operating system.
3. "The operating system tends to isolate the hardware from the users". Discuss this statement.
4. What is a supervisor ? How does it differ from the transient routines of an operating system ?
5. Differentiate between the terms throughput and turnaround time.
6. Explain how jobs are processed in batch mode.
7. What are the advantages and disadvantages of batch processing ?
8. What are some of the reasons that JCL



statements are needed when you submit your job for computer processing ?

9. Would you say that all computers use the same types of JCL statements ? Why ?
10. You want to compile and execute a COBOL program. In plain English, list out the necessary JCL statements you will prepare for this job.
11. What is spooling ? How does it help in improving the efficiency of a computer system ?
12. Define multiprogramming. Explain how multiprogramming ensures effective utilization of main memory and CPU.
13. Differentiate between I/O-bound and CPU-bound jobs.
14. List out some of the hardware and software facilities required for a multiprogramming system to work satisfactorily.
15. What is multiprocessing ? Give the basic organization of a multiprocessing system.
16. How is multiprocessing different from mutiprogramming ?
17. Discuss the advantages and limitations of mutiprocessing systems.
18. What is a time-slice ? In a timesharing system, explain how each and every user feels that he is the sole user of the computer system.
19. What are the three different states in which all users of a timesharing system fall ? Illustrate how a particular user switches from one state to another.
20. What is swapping ? How does it help an operating system in memory management ?
21. Why are timesharing systems considered to be most suitable for program development and testing ?
22. What type of hardware facilities are required for a timesharing computer system ?
23. Discuss the advantages and disadvantages of a timesharing system.
24. Multiprogramming and timesharing both involve multiple users in the computer concurrently. What is the basic difference between the two concepts ?
25. What is meant by real-time processing ? Give some examples of real-time applications.
26. "A real-time system uses on-line processing but an on-line system need not necessarily operate in real-time mode." Explain.
27. Why is a high degree of reliability necessary for real-time systems ? How is this achieved ?
28. Will it be practical to use magnetic tape files in conjunction with a real-time system? Give reasons for your answer.
29. What do you understand by the term "response-time" ? Why is response time critical in a real-time system ?
30. What is a virtual memory ? How is it implemented ?
31. What are the two main advantages of a virtual memory system ?
32. What are library routines ? Why are they normally supplied by the computer manufacturers ?
33. What is a librarian? What are its functions ?
34. What is a text editor ?
35. How is a dynamic debugging tool used by a programmer ?
36. Explain how memory dump programs and trace routines help a programmer in finding out mistakes in his program.
37. What is the use of peripheral interchange programs ?
38. One of your friends wishes to use your account in the computer to enter and test his program. What information you must provide for him/her to proceed ?



## CHAPTER 15

# 15. BUSINESS DATA PROCESSING CONCEPTS

This chapter deals with the basic concepts of business data processing. In this chapter, you will first learn the difference between data and information and the hierarchy of data storage. Then you will learn about the various types of file organizations and file utilities commonly used in business data processing applications. Finally, this chapter also introduces the basic concepts of data base systems and its advantages and limitations.

### WHAT IS DATA PROCESSING

*Data* are a collection of facts - unorganized but able to be organized into useful information. A collection of sales orders, time sheets, and class registration cards are a few examples. Data are manipulated to produce output, such as bills and paychecks. When this output can be used to help people make decisions, it is called *information*.

*Processing* is a series of actions or operations that convert inputs into outputs. When we speak of data processing, the input is data, and the output is useful

information. Hence, data processing is defined as series of actions or operations that converts data into useful information. The *data processing* system is used to include the resources such as people, procedures, and devices that are used to accomplish the processing of data for producing desirable output.

Thus, data are the raw material of information and just as raw materials are transformed into finished products by a manufacturing process, raw data are transformed into information by data processing.

### DATA STORAGE HIERARCHY

The basic building block of data is a *character*, which consists of letters (A, B, C...Z), numeric digits (0, 1, 2...9) or special characters (+, -, /, \*, ., \$...). These characters are put together to form a *field* (also called a fact, data item, or data element). A *field* is a meaningful collection of related characters. It is the smallest logical data entity that is treated as a single unit in data processing.



For example, if we are processing employees data of a company, we may have an employee code field, an employee name field, an hours worked field, an hourly-pay-rate field, a tax-rate-deduction field, etc. Fields are normally grouped together to form a record. A *record*, then, is a collection of related fields that are treated as a single unit. An employee record would be a collection of fields of one employee. These fields would include the employee's code, name, hours-worked, pay-rate, tax-rate-deduction, and so forth. Records are then grouped to form a file. A *file* is a number of related records that are treated as a unit. For example, a collection of all employee records for one company would be an employee file. Similarly, a collection of all inventory records for a particular company forms an inventory file. Figure 15.1 reveals these data relationships.

It is customary to set up a *master file* of permanent (and, usually, the latest) data, and to use *transaction files* containing data of a temporary nature. For example, the master payroll file will contain not only all the permanent details about each employee, his name and code, pay-rate, income tax rate and so forth, but it will also include the current gross-pay-to-date total and the tax paid-to-date total. The transaction payroll file will contain details of hours worked this week, normal and overtime, and, if piecework is involved, the quantity of goods made. When the payroll program is processed, both files will have to be consulted to generate this week's payslips, and the master file updated in readiness for the following week.

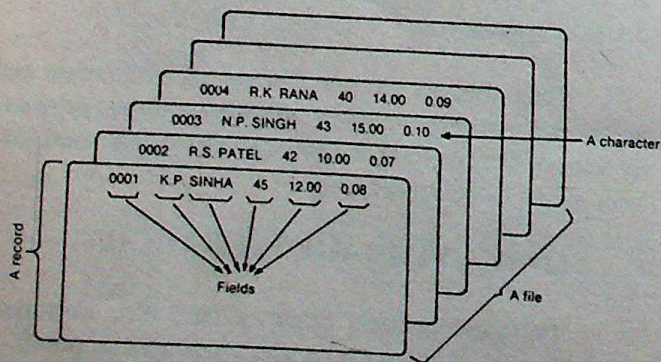


Figure 15.1. Relationship between character, field, record, and file.

A *data base* is a collection of integrated and related master files. It is a collection of logically related data elements that may be structured in various ways to meet the multiple processing and retrieval needs of organizations and individuals. Characters, fields, records, files, and data bases form a hierarchy of data storage. Figure 15.2 summarizes the data storage hierarchy used by computer-based processing systems. Characters are combined to make a field, fields are combined to make a record, records are combined to make a file, and files are combined to make a data base.

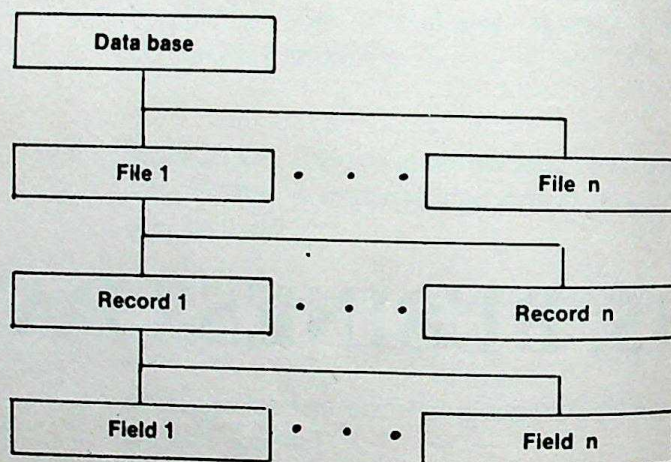


Figure 15.2. A data storage hierarchy.

## FILE ORGANIZATIONS

System designers choose to organize, access, and process records and files in different ways depending on the type of application and the needs of users. The three commonly used file organizations used in business data processing applications are - sequential, direct and indexed sequential organizations. The selection of a particular file organization depends upon the type of application. The best organization to use in a given application is the one that happens to meet the user's needs in the most effective and economical manner. In making the choice for an application, designers must evaluate the distinct strengths and weaknesses of each file organization. File organization requires the use of some *key field* or unique identifying value that is found in every record in the file. The key value must be unique for each record of the file because



duplications would cause serious problems. In the payroll example, the employee code field may be used as the key field.

### SEQUENTIAL FILES

In a sequential file, records are stored one after another in an ascending or descending order determined by the key field of the records. In payroll example, the records of the employee file may be organized sequentially by employee code sequence. Sequentially organized files that are processed by computer systems are normally stored on storage media such as magnetic tape, punched paper tape, punched cards, or magnetic disks. To access these records, the computer must read the file in sequence from the beginning. The first record is read and processed first, then the second record in the file sequence, and so on. To locate a particular record, the computer program must read in each record in sequence and compare its key field to the one that is needed. The retrieval search ends only when the desired key matches with the key field of the currently read record. On an average, about half the file has to be searched to retrieve the desired record from a sequential file.

#### Advantages of sequential files

1. Easy to organize, maintain, and understand.
2. There is no overhead in address generation. Locating a particular record requires only the specification of the key field.
3. Relatively inexpensive I/O media and devices can be used for the storage and processing of such files.
4. It is the most efficient and economical file organization in case of applications in which there are a large number of file records to be updated at regularly scheduled intervals. That is, when the *activity ratio* (the ratio of the total number of records in transaction file and the total number of records in master file) is very high. Applications such as payroll processing, billing and statement preparation, and bank cheque processing meet these conditions.

#### Disadvantages of sequential files

1. It proves to be very inefficient and uneconomical for applications in which the activity ratio is very low.
2. Since an entire sequential file may need to be

read just to retrieve and update few records, accumulation of transactions into batches is required before processing them.

3. Transactions must be sorted and placed in sequence prior to processing.
4. Timeliness of data in the file deteriorates while batches are being accumulated.
5. Data redundancy is typically high since the same data may be stored in several files sequenced on different keys.

### DIRECT FILES

A direct file (also called a random or relative file) consists of records organized in such a way that it is possible for the computer to directly locate the key of the desired record without having to search through a sequence of other records. This means that the time required for online enquiry and updating of a few records is much faster than when batch techniques are used. However, a direct-access storage device (DASD) such as a drum, disk, strip file, or mass core is essential for storing a direct file.

A record is stored in a direct file by its key field. Although it might be possible to directly use the storage location numbers in DASD as the keys for the records stored in those locations, this is seldom done. Instead, an arithmetic procedure called *hashing* is frequently used. In this method, an address generating function is used to convert the record key number into a DASD storage address. The address generating function is selected in such a manner that the generated addresses should be distributed uniformly over the entire range of the file area and a unique address should be generated for each record key. However, in practice, the above constraints are usually not satisfied and the address generating function often maps a large number of records to the same storage address. Several methods are followed to overcome this problem of *collision* when it occurs. One approach is to include a pointer field at the location calculated by the hashing function. This field points to the DASD location of another record that has the same calculated address value. When the computer is given the key of a record to be processed at a later date, it reuses the hashing function to locate the stored record. If the record is found at the location calculated by the hashing function, the search is over and the record is directly accessed for processing. On the other hand, if the record at the calculated address does not have the correct key, the computer looks at the pointer field to continue the search.



### Advantages of direct files

1. The access to, and retrieval of a record is quick and direct. Any record can be located and retrieved directly in a fraction of a second without the need for a sequential search of the file.
2. Transactions need not be sorted and placed in sequence prior to processing.
3. Accumulation of transactions into batches is not required before processing them. They may be processed as and when generated.
4. It can also provide up-to-the minute information in response to inquiries from simultaneously usable online stations.
5. If required, it is also possible to process direct file records sequentially in a record key sequence.
6. A direct file organization is most suitable for interactive online applications such as airline or railway reservation systems, teller facility in banking applications, etc.

### Disadvantages of direct files

1. These files must be stored on a direct-access storage device. Hence, relatively expensive hardware and software resources are required.
2. File updation (addition and deletion of records) is more difficult as compared to sequential files.
3. Address generation overhead is involved for accessing each record due to hashing function.
4. May be less efficient in the use of storage space than sequentially organized files.
5. Special security measures are necessary for online direct files that are accessible from several stations.

### INDEXED SEQUENTIAL FILES

We are all familiar with the concept of an index. For example, the directory in a large multistoried building is an index that helps us to locate a particular person's room within the building. For instance, to find the room of Dr. Sharma within the building, we could look up his name in

the directory (index) and read the corresponding floor number and room number. This idea of scanning a logically sequenced table is preferable to searching door by door for the particular name. Similarly, if we wished to read the section in this book about printers, we would not begin on page 1 and read every page until we came across the topic of interest. Rather, we would find the subject in the contents (which serves as an index) to locate the page number, and then turn directly to that page to begin reading.

Indexed sequential files use exactly the same principle. The records in this type of file are organized in sequence and an index table is used to speed up access to the records without requiring a search of the entire file. The records of the file can be stored in random sequence but the index table is in sorted sequence on the key value. This provides the user with a very powerful tool. Not only can the file be processed randomly, but it can also be processed sequentially. Since the index table is in a sorted sequence on the key value, the file management system simply accesses the data records in the order of the index values. Thus indexed sequential files provide the user sequential access, even though the file management system is accessing the data records in a physically random order.

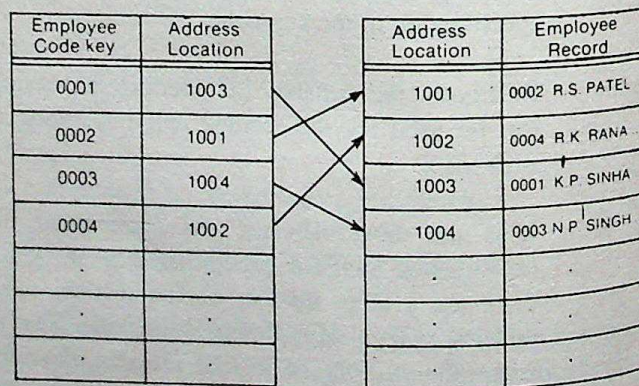


Figure 15.3. Organization of an indexed sequential file.

This concept is illustrated in Figure 15.3. This technique of file management is commonly referred to as the Indexed Sequential Access Method (ISAM). Files of this type are called ISAM files.



### Advantages of indexed sequential files

1. Permits the efficient and economical use of sequential processing techniques when the activity ratio is high.
2. Permits direct access processing of records in a relatively efficient way when the activity ratio is low.

### Disadvantages of indexed sequential files

1. These files must be stored on a direct-access storage device. Hence, relatively expensive hardware and software resources are required.
2. Access to records may be slower than direct files.
3. Less efficient in the use of storage space than some other alternatives.

## FILE UTILITIES

File utilities consist of routines which perform a variety of generalised operations on data files. Normally, file utilities are data independent. This means that the routines are written quite generally, and will operate on any data formats and even on data held on different types of storage medium. Some of the commonly used file utilities are discussed below.

### SORTING

The purpose of sorting a file is to arrange records within a file in some defined sequence. This sequence is determined by the ordering of certain specified fields within the record. Fields whose ordering determine the sequence of a file in the sorting process are known as *keys*. The simplest case is an ordering on a single key. For example, a file of personnel records may be sequenced by ascending order of employee code as shown in Figure 15.4. A more complex ordering may be produced by introducing a further key in the sorting process. For example, suppose each record of the personnel file also contains a field for department number to which the employee belongs. Now the order of sorting may be employee code within department number. This means that all records for the lowest department number are presented first, each in

| Employee code | Department Number |
|---------------|-------------------|
| 101           | 2                 |
| 123           | 3                 |
| 124           | 1                 |
| 176           | 2                 |
| 178           | 1                 |
| 202           | 3                 |
| 213           | 1                 |

Figure 15.4. Sorting on one key in ascending employee code sequence.

| Employee Code | Department Number |
|---------------|-------------------|
| 124           | 1                 |
| 178           | 1                 |
| 213           | 1                 |
| 101           | 2                 |
| 176           | 2                 |
| 123           | 3                 |
| 202           | 3                 |

Figure 15.5. Sorting on two keys. Ascending employee code within ascending department number. Department number is primary key and employee code is secondary key.

ascending sequence of employee code : then all records for the next department number and so on. This is indicated in Figure 15.5. In this example, two keys have been used in the sorting process - department number is called the *primary key* and employee code is known as the *secondary key*.

According to the extent and sophistication of the sort utility available, the size and number of keys which can be specified, and the type of ordering (e.g. ascending, descending, alphabetical) will vary.



Since sorting is a very common data-processing requirement, manufacturers provide sort utility software which enables users to specify their particular sequencing requirements by means of simple parameters. Software is usually available for sorting files held on all types of storage devices. The user specifies the sort keys, and also details about the type of file such as storage device, file labels, record structure. The sort utility program reads the unsequenced input file, and by means of various copying techniques ultimately produces as output a copy of the input file in the required sequence.

### SEARCHING

Searching is the process of scanning a file to find a particular record. The efficiency of a search algorithm depends on the file organisation. For example, to search a particular record in a sequential file, the file is scanned sequentially beginning with the first record and the desired key is compared one-by-one with the key field of each record. The search process terminates when the matching key is found. On the other hand, direct file organisation enables the program to have immediate access to the desired record. The program need only inform the file management system which record is needed, and the management system then searches through the filing system and produces the desired record. Normally, the time required to search a particular record from a direct file is much less as compared to the time required to search it from a sequential file.

### MERGING

Merging of files involves the combining of records from two or more ordered files into a single ordered file. Each of the constituent files must be in the same order, although the record layout of files need not be identical. The output file will be in the same order as the input files, placing records from each in their correct relative order.

For example, in Figure 15.6, files A and B are merged to produce an output file C.

### COPYING

File copying routines are provided for producing an exact copy of a file, either from one unit of a storage device onto another similar unit, e.g., from one tape reel to another, or from one storage medium to another, e.g., copying a card or disk file onto tape.

These routines are normally used for taking back-up copies of useful files. For example, a disk file may be copied on a tape or floppy for back-up purpose. The

copying routines are also known as peripheral interchange programs since they are used to copy a file from one peripheral device onto another peripheral device.

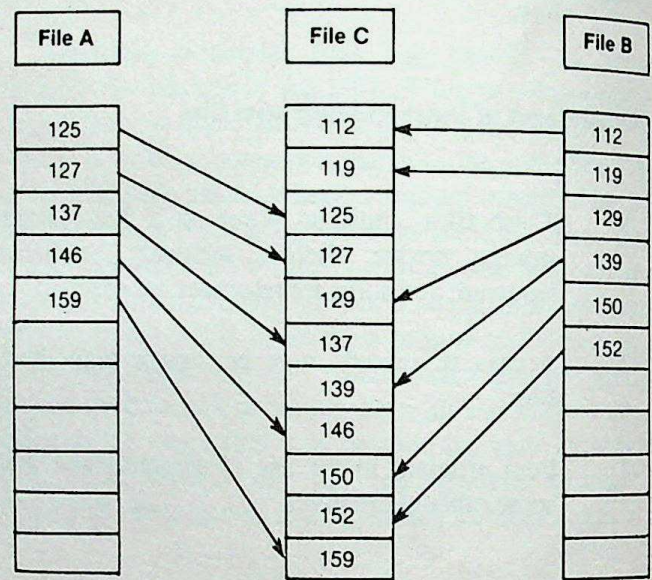


Figure 15.6. Merging of files A and B to produce file C.

### PRINTING

Printing routines are used to print file contents upon a printer. File contents may be reproduced in different formats on the printer if there is a difference in the internal and external data formats, e.g. binary to character conversion. Special printing may be provided if the file contains program instructions rather than data. Some selection and editing facilities are normally provided with printing routines to enable parts of files to be output, e.g. specified number of records or blocks and certain portions only of records to be printed.

### MAINTENANCE

File maintenance is the term given to any system of reorganisation of data items within a file, where the reorganisation is independent of the information content of the file. File maintenance software in effect is a form of



today. However, for most modern processing needs, this type of system has a number of shortcomings, which are as follows:

1. *Data redundancy.* The same basic data fields are included in many different files. For example, suppose a file containing employee details is organized according to the employee code as key field. If we wish to get the names of employees with a certain educational qualification from this file, we need to search all file records. If the need for such an information became a routine, then a new file structured on an educational background key would be created and a new program would be written to process this file. Of course, this second file would duplicate much of the data stored in the first personnel file.
2. *Data inconsistency.* When changes occur in a data item, every file which contains that field should be updated to reflect the change. Confusion can result when one file is updated while another file containing the same field is not updated. This is a very common occurrence and leads to the problem of inconsistency. If one of the files is inadvertently not updated correctly, then the same entries in different files will be different (inconsistent) resulting in frequent discrepancies among reports produced from different files.
3. *Lack of program/data independence.* The programs used with file oriented applications usually contain "picture," "format," or "data" statements that precisely define each data field to be processed. Anytime if there is a need to add, delete, or change data formats, the application program must also be changed. Likewise, a significant revision in a program may require a restructuring of the data file processed by the program. Changing programs to accommodate data format changes is a major maintenance activity in many data processing installations today.
4. *Wastage of resources.* Redundancy of data between files also results in wasted storage and wasted processing time in updating all files. If the same data item is stored in several files, obviously wasted storage will result. Moreover, the cost of entering and storing the same data in many files can be quite expensive.



## THE DATA BASE CONCEPT

In order to overcome the problems of a conventional file-oriented data processing system, the data base concept was introduced. Although there are differences of opinion about what constitutes a data base system, the most prevalent view is that such systems possess the following characteristics :

1. It is a centralized and integrated shared data file which consists of all data used by a company.
2. It is organized and structured in a different manner than the conventional sequential file organizations.
3. Its organization permits access to any or all data quantities by all applications with equal ease.
4. Its organization is such that duplication of data is minimised if not eliminated entirely.
5. It emphasizes the independence of programs and data. It involves the concept of separating data definition from the applications programs and including it as part of the database.
6. It provides for the definition of logical relationships which exist between various records in the data base.
7. It is stored on a direct-access storage device.

The first step in moving from ordinary file management to a data base system is to separate all data definitions from the applications programs and to consolidate them into a separate entity called a *schema*, as illustrated in Figure 15.7. In addition to data definition, the schema also includes an indication of the logical relationships between various components of the data base. This is represented in Figure 15.7 by the data structure definitions. In other words, virtually everything there is to know about the data base and its structure is included in the schema.

The schema then becomes a component of the overall data base itself. From the schema the installation can generate dictionaries containing a complete description of the data base. These will, in turn, be used by systems analysts in defining new applications.

Data base systems are typically installed and coordinated by an individual called the *data base administrator*. He has the overall authority to establish and

control data definitions and standards. He is responsible for determining the relationships among data elements, and for designing the data base security system to guard against unauthorised use. He also trains and assists applications programmers in the use of data base. A data dictionary is developed and used in a data base to document and maintain the data definitions.

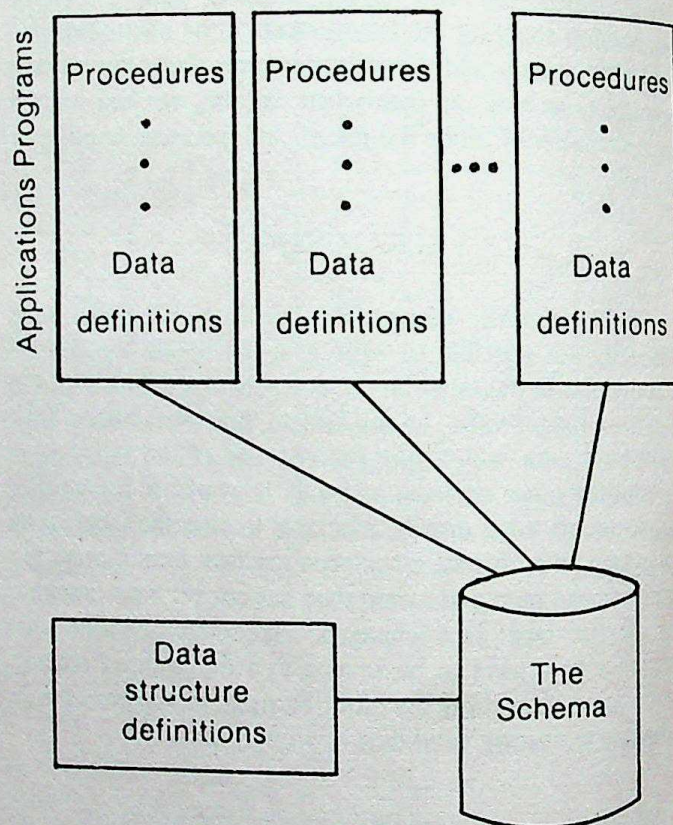


Figure 15.7. Illustrating the use of schema in data base for separating data definitions from programs.

## DATA BASE MANAGEMENT SYSTEM

A collection of programs required to store and retrieve data from a data base is called a *data base management system (DBMS)*. As shown in Figure 15.8, the principal components of a DBMS are a data description module and a data manipulation module. The data description module of the DBMS analyzes the data requirements of applications programs and transfers control to the data manipulation module, which retrieves the needed data elements from the data base.



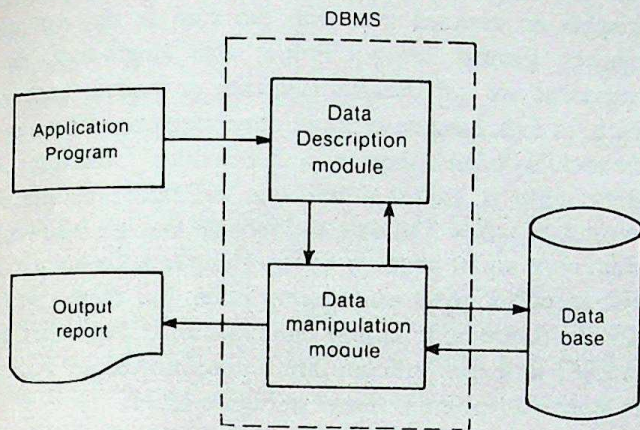


Figure 15.8. Principal components of a DBMS.

A DBMS can organize, process, and present selected data elements from the data base. This capability enables decision makers to search, probe, and query data base contents in order to extract answers that are not available in regular reports. For example, a query of the form "list out all male employees who are more than 45 years old and less than 50 years old and whose basic salary is more than Rs 3000/- per month" can easily be answered from an employee data base.

Data base management systems free the programmer from the need to worry about the organization and location of data. All of the data needed by an application program can be accessed, regardless of access method, record location, or record content. Programming is speeded up because the programmer can concentrate upon the logic of the application. Most DBMS are designed to interact with the commonly used programming languages such as COBOL. Many DBMS include special, user-friendly *query languages*. These languages can be easily learned by nonprogramming users of the system, enabling them to access the data base for information as needed without the help of any programmer.

### DATA BASE STRUCTURING TECHNIQUES

We have seen how sequential, direct and indexed sequential approaches are used to organize and structure the data in *single* files. However a DBMS is able to integrate data elements from several files to answer specific user inquiries for information. This means that the DBMS is

able to access and retrieve data from nonkey record fields. That is, the DBMS is able to structure and tie together the logically related data from several large files. Identifying these logical relationships is a job of the data base administrator. A *data definition language* is used for this purpose. The DBMS may then employ one of the following structuring techniques for the efficient storage, access, and retrieval operations of data.

**List structures.** In this method, records are linked together by the use of pointers. A *pointer* is a data item in a record that identifies the storage location of another logically related record. For example, let us assume that an employee file contains five records. There is one record for each employee. Now assume that we wish to find out all employees with a job classification of C2. One way to do this is to add a pointer at the end of each record for job classification C2. The pointer simply indicates (or points to) the address or location of next record containing an employee with a job classification of C2. Pointers for job classification C2 are shown in Figure 15.9. When a list of records are tied together using pointers, the list is called a *linked list*.

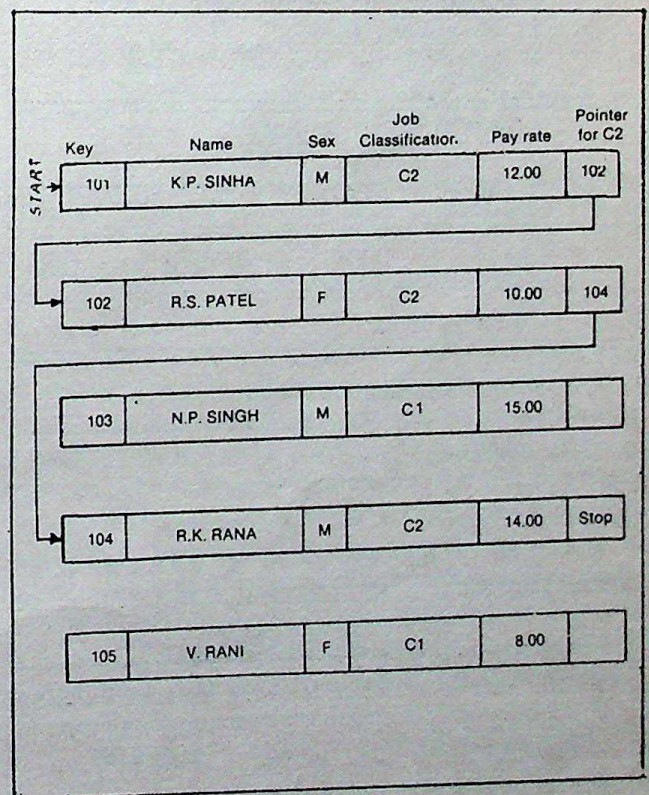


Figure 15.9. A list structure using a pointer for C2.



With pointer, the addresses and locations of related records are actually placed in the records themselves. An alternative to pointers is the index or inverted list. In this method, the relationship between various data elements is placed in a separate file or table called an *index* or an *inverted list*. It is referred to as an inverted list because the attribute values, such as sex, job classification, and pay rate are inverted with the keys used for direct access. In other words, the key for the record and the actual contents or attributes of the record are reversed. This allows us to start with an attribute, such as job classification, and determine which records contain employees with that particular attribute. Inverted lists for sex, job classification, and pay rates of employees of Figure 15.9 are shown in Figure 15.10.

Suppose if there is a query to determine all employees belonging to job class C1, then the computer goes to the job classification index and reads the pointer values for job classification C1. This allows the computer to directly access the records of employees 103 and 105, the employees belonging to job classification C1. Similarly, the index for sex can be used to list out all the male or all the female employees directly. Also the pay rate index can be used to find employees with certain pay rate ranges.

| SEX INDEX                |                |
|--------------------------|----------------|
| Key Values               | Pointer Values |
| M                        | 101, 103, 104  |
| F                        | 102, 105       |
| JOB CLASSIFICATION INDEX |                |
| Key Values               | Pointer Values |
| C1                       | 103, 105       |
| C2                       | 101, 102, 104  |
| PAY RATE INDEX           |                |
| Key Values               | Pointer Values |
| 5.01–10.00               | 102, 105       |
| 10.01–15.00              | 101, 103, 104  |

Fig 15.10. Example of indexes and inverted lists.

**Hierarchical or tree structures.** In this method, data units are structured in multiple levels that graphically resemble an inverted tree with the root at the top and branches formed below. Below the single-root data component are subordinate elements or nodes, each of which, in turn, has one or more other elements. There is a parent-child relationship in a hierarchical structure. A parent node is one that has one or more subordinate elements or nodes. The data and records that are below the parent node are its children nodes. There may be numerous children nodes under each parent node, but there can be only one parent node for any one child node. Note that the branches in a tree structure are not connected. A typical hierarchical structure is shown in Figure 15.11.

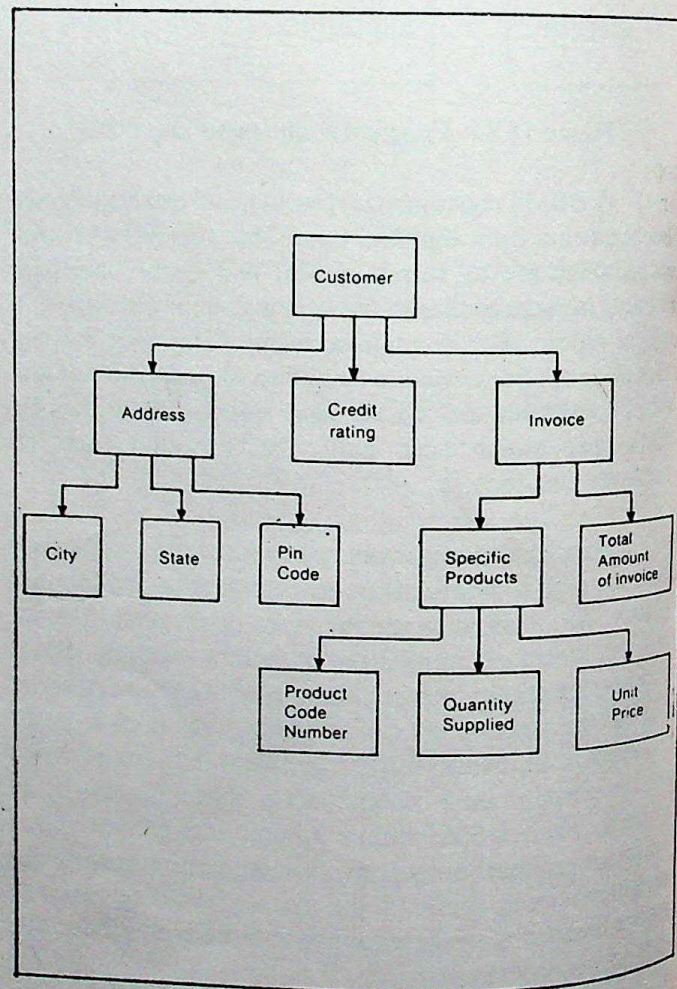


Figure 15.11. A typical hierarchical (tree) structure.



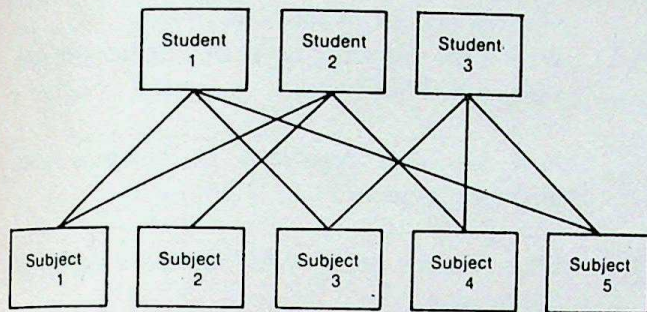


Figure 15.12. A typical network structure.

The hierarchical structure for data bases is very popular. A large number of computer programs have been written that use this structure. On the other hand, there are also disadvantages. While it is possible to force most data into a hierarchical structure, it can be difficult and awkward. In addition, a hierarchical structure can be more complex and more difficult to understand for managers and executives. Even then, the hierarchical structure continues to be a dominant one for large organizations, such as airline companies and credit card companies.

*Network structures.* A network structure is an extension of the hierarchical or tree structure. Instead of having only one parent node, however, the network structure can have multiple parent nodes for a child node. Thus, each node may have several owners and may, in turn, own any number of other data items. Data management software permits the extraction of the needed information from such a structure by beginning with any record in a file.

A typical network structure illustrating the relationship between students and the subjects taken by them is shown in Figure 15.12. There are three students and five subjects. The lines drawn between the students and the subjects show which students have taken which subjects. It can be easily seen that student 1 has taken subjects 1, 3, and 5, that student 2 has taken subjects 1, 2, and 4, and that student 3 has taken subjects 3, 4, and 5. Of course, there could be thousands of students and many more subjects.

**Relational structures.** The overall purpose of this model is to relate data records using a standard tabular format. It uses normal two-dimensional tables to describe all relationships between data. For example, relational

tables could be established to link a college course with the instructor of the course, and with the location of the class as shown in Figure 15.13. To find the name of the instructor and the location of the Hindi class, the course/instructor relation is searched to get the name of the instructor (R. Pandey), and the course/location relation table is searched to get the class location (Room 210). Many other relations are, of course, possible.

| COURSE/INSTRUCTOR<br>RELATION TABLE |            |
|-------------------------------------|------------|
| Course                              | Instructor |
| English                             | S.K. Ray   |
| Hindi                               | R. Pandey  |
| Physics                             | P.K. Sen   |
| Chemistry                           | R.S. Gupta |
| Maths                               | N.P. Singh |
| •                                   | •          |
| •                                   | •          |

| COURSE/LOCATION<br>RELATION TABLE |          | OTHER<br>RELATION TABLES                                                                                           |
|-----------------------------------|----------|--------------------------------------------------------------------------------------------------------------------|
| Course                            | Location |                                                                                                                    |
| English                           | Room 105 | For example<br>tables relating<br>courses with<br>time of meeting,<br>days of meeting,<br>hours of credit,<br>etc. |
| Maths                             | Room 203 |                                                                                                                    |
| Chemistry                         | Room 115 |                                                                                                                    |
| Physics                           | Room 108 |                                                                                                                    |
| Hindi                             | Room 201 |                                                                                                                    |
| •                                 | •        |                                                                                                                    |
| •                                 | •        |                                                                                                                    |

Figure 15.13. Illustrating relational structure.

The relational model is a relatively new data base structuring technique. This model does not force us to use a structure, such as the hierarchical structure or the network structure. This is one of the advantages of the relational approach. The data in the relational model, in most cases, can be identical to the actual relationships that exist between the records and various data items. This is an important advantage over hierarchical and network models. As a result, there has been a considerable amount of interest in developing relational data base systems. Today relational models exist for large computers as well as small microcomputers.

## ADVANTAGES AND LIMITATIONS OF DATA BASE SYSTEMS

In comparison to a conventional file-oriented data processing system, a data base system enjoys the following advantages :



1. The integration and sharing of data files minimizes the duplication and redundancy of data to a great extent.
2. Integration of data files also results in a considerable saving of storage space and in data entry and data storage costs.
3. Fewer applications programs need to be developed for obtaining various reports due to independence of programs and data.
4. The query language facility helps non-programming personnel to access the data base for information as needed without the help of any programmer.
5. Faster preparation of information to support nonrecurring tasks and changing conditions is possible.
6. Updation of data becomes easier due to integration of data files. Fewer errors may result when several records may be updated simultaneously.
4. Describe the data storage hierarchy.
5. Give an example to illustrate the relationship between a character, a field, a record, and a file.
6. What is the difference between a master file and a transaction file ?
7. How are data organized in business data processing systems ?
8. What is a key field ? What role does it play in file creation ?
9. How is a sequential file organized ? How are records in a sequential file accessed ? How are these records processed ?
10. What conditions support the use of sequential file ?
11. How is a record stored in a direct file ? How is it retrieved and processed ?
12. What conditions support the use of direct files and direct-access processing ?
13. How are records stored in an indexed sequential file ? How are they retrieved and processed ?
14. Discuss the advantages and limitations of the sequential, direct, and indexed sequential file approaches.
15. What are file utilities ?
16. Differentiate between the process of sorting and merging of files.
17. Explain the use of the following file utilities : copying, printing, labelling, and scratching.
18. What are the shortcomings of a conventional file oriented data processing system ?
19. What are the characteristics of a data base system ?
20. What is a data base schema ?
21. What are the jobs and responsibilities of a data base administrator ?
22. What is a data dictionary ?

On the other hand, the following are some of the limitations of a data base system :

1. More complex and expensive hardware and software resources are needed.
2. Sophisticated security measures must be implemented to prevent unauthorised access of sensitive data in online storage.
3. Hardware or software failures might result in the destruction of vital data base contents.
4. A lengthy conversion period may be needed, higher personnel training costs may be incurred, and more sophisticated skills are needed by those responsible for the data base system.

## QUESTIONS

1. What is the difference between data and information ?
2. What is meant by data processing ?
3. What is a data processing system ?
4. What is a data base ?
5. What is a data base system ?
6. What is a data base administrator ?
7. What is a data dictionary ?
8. What is a data base schema ?
9. What are the jobs and responsibilities of a data base administrator ?
10. What is a data dictionary ?
11. What is a data base system ?
12. What is a data base schema ?
13. What are the jobs and responsibilities of a data base administrator ?
14. What is a data dictionary ?
15. What is a data base system ?
16. What is a data base schema ?
17. What are the jobs and responsibilities of a data base administrator ?
18. What is a data dictionary ?
19. What is a data base system ?
20. What is a data base schema ?
21. What are the jobs and responsibilities of a data base administrator ?
22. What is a data dictionary ?



23. What is a data base management system ? How is it used ?
24. Identify and discuss the structuring techniques used by data base management systems.
25. What is an inverted list ? How does it compare to a pointer ?
26. What is the difference between a hierarchical and a network structure ?
27. What are the advantages and limitations of a data base system ?







## CHAPTER 16

# 16. DATA COMMUNICATIONS AND COMPUTER NETWORKS

With the miniaturization of hardware components and the drastic reduction in their cost, computer hardware has evolved from an item of major economic concern with first generation digital computers to one of minor concern in today's computing systems. The impact of this recession was a move from mainframes and minicomputers to small personal computers. The simultaneous advancements in data communications facilities have increased the usability of remote computing systems and have led to their consideration as alternatives to centralized data processing. With earlier computer systems, all components of the computer system were typically located in one or two rooms. Today computer systems are literally reaching out to us. Instead of taking processing jobs to the computer room, we are now bringing the computer systems to where the action is.

introduce the various aspects of data communications technology and to discuss how this technology is combined with computer technology to form computer networks. Following are some of the important concepts which you will learn from this chapter:

1. Basic elements of a communication system.
2. The techniques, channels and devices used to transmit data between distant locations.
3. The various types of computer networks.
4. Communication protocols which are rules and procedures for establishing and controlling the transmission of data from one computer to another.

The marriage of computers with data communications is one of the most exciting developments in data processing today. The goal of this chapter is to

Characteristics and advantages of distributed data processing.



## BASIC ELEMENTS OF A COMMUNICATION SYSTEM

Communications is the process of transferring messages from one point to another. As shown in Figure 16.1, the three basic elements of any communications process are :

1. A *sender* (source) which creates the message to be transmitted.
2. A *medium* which carries the message.
3. A *receiver* (sink) which receives the message.

For example, when you speak to your friend on the telephone, you are the sender. The telephone line through which your voice is transmitted is the medium and your friend is the receiver. This is a simple example of voice communication. The same concept holds good for data communications also. Data communications is the function of transporting data from one point to another. In this case, the sender and the receiver are normally machines, in particular, computer devices (computers, terminals, peripheral devices like line printers,

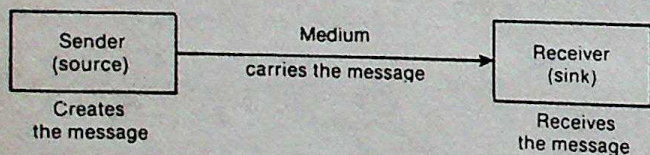


Figure 16.1. Basic elements of a communication system.

teletypes, disks, etc.) and the transmission medium may be telephone lines, microwave links, satellite links, etc. However, the messages that are transmitted are data, not voice conversations. Thus, the electronic systems that transfer data from one point to another are called *data communications systems*. Unlike computers that process and rearrange data, data communications systems transmit data from one point to another without any change.

## DATA TRANSMISSION MODES

There are three ways, or modes, for transmitting data from one point to another. As shown in Figure 16.2, these are simplex, half-duplex and full-duplex.

### SIMPLEX

If transmission is simplex, communication can take place in only one direction. Devices connected to such a circuit is either a send-only or a receive-only device. For example, a data collection terminal on a factory floor (send only) or a line printer (receive only). At first thought that might appear adequate for many types of applications in which

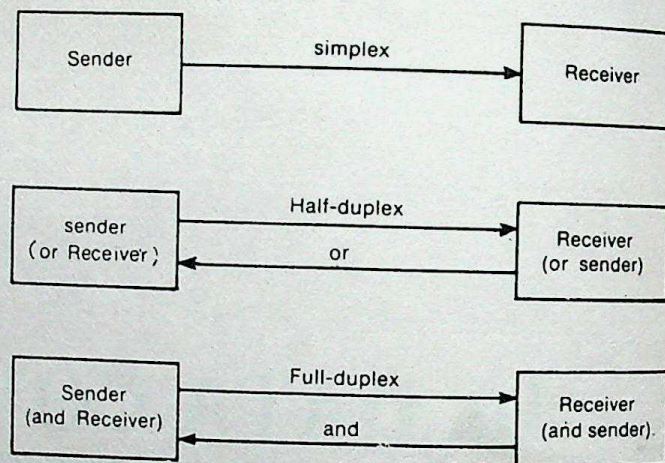


Figure 16.2. Modes of transmission.

flow of information is unidirectional. However, in almost all data processing applications, communication in both directions is required. Even for a "one-way" flow of information from a terminal to a computer, the system will be designed to allow the computer to signal the terminal that data has been received. Without this capability, the remote user might enter data and never know that it was not received by the computer (due to some problem somewhere). Hence, simplex circuits are seldom used because a return path is generally needed to send acknowledgement, control, or error signals.

### HALF-DUPLEX

A half-duplex system can transmit data in both directions, but only in one direction at a time. Thus, a half-duplex line can alternately send and receive data. It requires two wires. This is the most common type of transmission for voice communications because only one person is supposed to speak at a time. It is also used to connect a terminal with a computer. The terminal might transmit data and then the computer responds with an acknowledgement.



## FULL-DUPLEX

In a half-duplex system, the line must be "turned-around" each time the direction is reversed. This involves a special switching circuit and requires a small amount of time (approximately 150 milliseconds). With high speed capabilities of the computer, this turn-around time is unacceptable in many instances. Also, some applications require simultaneous transmission in both directions. In such cases, a full-duplex system is used that allows information to flow simultaneously in both directions on the transmission path. Use of a full-duplex line improves efficiency as the line turn-around time required in a half-duplex arrangement is eliminated. It requires four wires.

## DATA TRANSMISSION SPEED

A term used to describe the data-handling capacity of a communications system is bandwidth. *Bandwidth* is the range of frequencies that is available for the transmission of data. A narrow range of frequencies in a communications system is analogous to a narrow road. The flow of information in such a system - its data transfer rate - is limited, just as is the flow of traffic in the narrow road. Wider bandwidths permit more rapid information flow. Thus, the wider the bandwidth of a communications system, the more data it can transmit in a given period of time.

When dealing with computer input/output devices we think in terms of characters per second. However, in case of data transmission, we talk in terms of bits per second. Recall that the ASCII code uses 7 data bits per character plus a parity bit. For data communication, additional bits are added to control the process. Although the number of bits depends upon the communications system used, commonly encountered systems use a total of either 10 or 11 bits per character. Hence a terminal having a speed of 30 characters per second would probably be used with a communications system which transmits at the rate of 300 bits per second.

The communication data transfer rate is measured in a unit called *baud*. In general usage, *baud* is identical to *bits per second*. For instance, a rate of 300 baud is 300 bits per second. Therefore, the 30 characters per second terminal would be said to operate at 300 baud. However, technically *baud* refers to the number of signal (state) changes per second. Thus, using more sophisticated coding techniques, 1 baud can represent 2 or even 3 bits per second. But, with most communications systems, 1 baud represents only one signal change per second and thus is equivalent to 1 bit per second.

Depending on their transmission speeds, communication channels (paths) are grouped into three basic categories - narrowband, voiceband, and broadband.

## NARROWBAND

Narrowband or subvoice grade channels range in speed from 45 to 300 baud. They are used for handling low data volumes and are adequate for low-speed devices. They are used mainly for telegraph lines and low speed terminals.

## VOICEBAND

Voiceband channels handle moderate data volumes and can transmit data at speeds upto 9600 baud. They are so called because their major application is for ordinary telephone voice communication. They are also used for data transmission from card reader to CPU or from CPU to line printer. Moreover, most remote terminals are connected to computers through voiceband channels.

## BROADBAND

Broadband or wideband channels are used when large volumes of data is to be transmitted at high speed. These systems provide data transmission rates of 1 million baud or more. A company might use a broadband facility for high speed computer to computer communication or for simultaneous transmission of data to several different devices.

As might be expected, the cost of the service increases with speed. Hence a thorough analysis of the business needs and associated costs is necessary in making a proper choice.

## TRANSMISSION MEDIA

The sender-medium-receiver concept has actually been with us for a very long time. For example, shouting to another person does in fact involve voice transmission over a distance via the medium air which carries the sound wave. Needless to say, the use of telephone lines as a transmission medium considerably enhances the possible distance. Like telephone lines, there are several types of physical channels (communication media) through which data can be transmitted from one point to another. Some of the most common data transmission medium are briefly described



## WIRE PAIRS

Wire pairs are commonly used in local telephone communication and short distance (upto about 1 km) digital data transmission. They are usually made of copper and a pair of wires are twisted together (Figure 16.3) to reduce interference by adjacent wires.

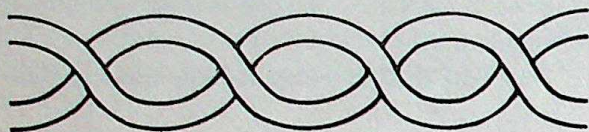


Figure 16.3. A wire pair.

Wire pairs are normally used to connect terminals to the main computer if they are placed at a short distance from the main computer. Data transmission speed of upto 9600 bits per second (or 9600 baud) can be achieved if the distance is not more than 100 meters. However, for longer distance, local telephone lines are used. In this case, typical speed of digital signal transmission is 1200 bits per second.

Wire pairs are an inexpensive medium of data transmission. They are easy to install and use. However, their use is limited because they easily pick up noise signals which results in high error rates when the line length extends beyond 100 meters.

## COAXIAL CABLE

Coaxial cables are groups of specially wrapped and insulated wire lines that are able to transmit data at high rates. As shown in Figure 16.4, they consist of a central copper wire surrounded by a PVC insulation over which a sleeve of copper mesh is placed. The metal sleeve is again shielded by an outer shield of thick PVC material. The signal is transmitted by the inner copper wire and is electrically shielded by the outer metal sleeve.

Coaxial cables offer much higher bandwidths and are capable of transmitting digital signals at very high rates of 10 mega bits per second. They are extensively used in long distance telephone lines and as cables for closed circuit TV. They are also used by telephone companies to transmit data. In many cases, coaxial cables are packaged into a very large cable that can handle over 15,000 telephone calls simultaneously. Furthermore, coaxial cables have much higher noise immunity and can offer cleaner and crisper data transmission without distortion or loss of signal.

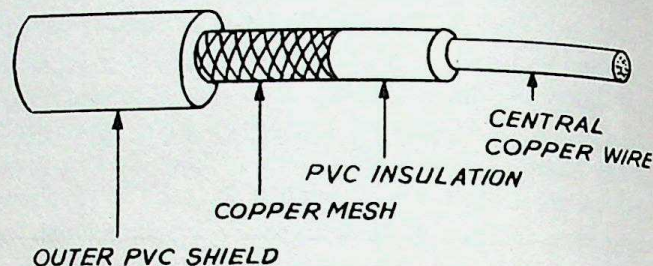


Figure 16.4. A coaxial cable.

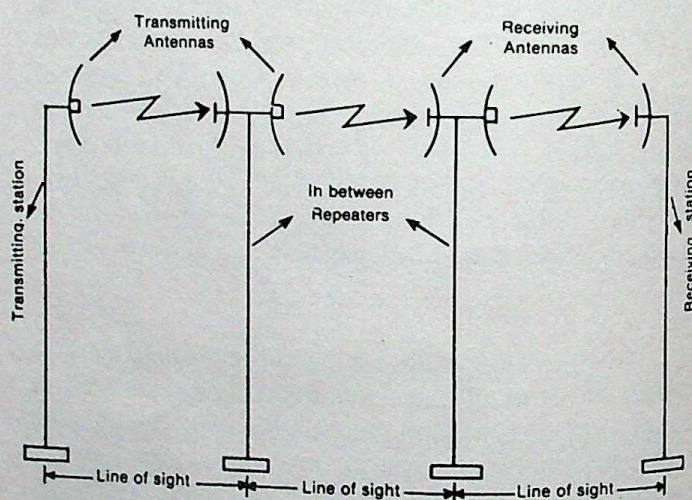


Figure 16.5. Illustrating microwave communication from one point to another.

## MICROWAVE SYSTEM

Another popular transmission media is microwave. This is a popular way of transmitting data since it does not incur the expense of laying cables. Microwave systems use very high frequency radio signals to transmit data through space. However, at microwave frequencies the electromagnetic waves cannot bend or pass obstacles like hill. Hence it is necessary that microwave transmission is in a line-of-sight. In other words, the transmitter and receiver



of a microwave system, which are mounted on very high towers, should be in a line-of-sight. This may not be possible for very long distance transmission. Moreover, the signals become weaker after travelling a certain distance and require power amplification.

In order to overcome the problem of line-of-sight and power amplification of weak signals, microwave systems use repeaters at intervals of about 25 to 30 kms inbetween the transmitting and receiving stations (see Figure 16.5). The first repeater is placed in line-of-sight of the transmitting station and the last repeater is placed in line-of-sight of the receiving station. Two consecutive repeaters are also placed in line-of-sight of each other. The data signals are received, amplified, and retransmitted by each of these stations.

Microwave systems permit data transmission rates of about 16 Giga (1 Giga =  $10^9$ ) bits per second. Furthermore it can carry literally thousands of voice channels at the same time. The link can support 250,000 voice channels. However, the capital investment needed to install microwave links is very high and hence they are mostly used to link big metropolitan cities with heavy telephone traffic between them.

### COMMUNICATIONS SATELLITE

The main problem with microwave communication is that the curvature of the earth, mountains, and other structures often block the line-of-sight. Due to this reason, several repeater stations are normally required for long distance transmission which increases the cost of data transmission between two points. This problem is overcome by using satellites which are relatively newer and more promising data transmission media.

A communication satellite is basically a microwave relay station placed in outer space. These satellites are launched either by rockets or by space shuttles and are precisely positioned 36000 kms above the equator with an orbit speed that exactly matches the earth's rotation speed. Since a satellite is positioned in a geosynchronous orbit, it is stationary relative to earth and always stays over the same point on the ground. This allows a ground station to aim its antenna at a fixed point in the sky. Dozens of satellites are now in orbit to handle international and domestic data, voice, and video communications needs. The Indian satellite, INSAT-1B, is positioned in such a way that it is accessible from any place in India.

As shown in Figure 16.6, in satellite communication, microwave signal at 6 GHz (read as gigahertz =  $10^9$  Hz) is transmitted from a transmitter on

earth to the satellite positioned in space. By the time this signal reaches the satellite it becomes weak due to 36000 kms travel. The satellite amplifies the weak signal and transmits it back to the earth at a frequency of 4 GHz. This signal is received at a receiving station on the earth. It may be noted that the retransmission frequency is different from the transmission frequency in order to avoid the interference of the powerful retransmission signal with the weak incoming signal.

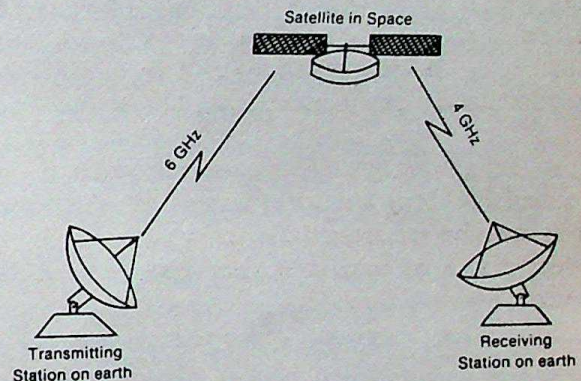


Figure 16.6. Illustrating satellite communication from one point to another.

The main advantage of satellite communication is that it is a single microwave relay station visible from any point of a very-very large area. For example, satellites used for national transmission are visible from any part of the country. Thus transmission and reception can be between any two randomly chosen places in that area. Moreover, transmission and reception costs are independent of the distance between two points. In addition to this, a transmitting station can receive back its own transmission and check whether the satellite has transmitted the information correctly. If an error is detected, the data would be retransmitted. However, a major drawback of satellite communications has been the high cost of placing the satellite into its orbit. Moreover, a signal sent to a satellite is broadcast to all receivers within the satellite's range. Hence necessary security measures are to be taken to prevent unauthorised tampering of information.



## OPTICAL FIBERS

A potential cost-effective earth alternative to satellite communications for the future will be the use of optical fibers. The Bell System of U.S.A. has tested small half-inch-diameter cables that could transmit as many as 50,000 voice messages simultaneously by means of advanced optical techniques based upon laser technology.

In fiber optics, semiconductor lasers transmit information in the form of light along hair-thin glass (optical) fibers at 186,000 miles per second (the speed of light), with no significant loss of intensity over very long distances. As shown in Figure 16.7, the system basically consists of fiber optic cables that are made of tiny threads of glass or plastic. Towards its source side is a converter that converts electrical signals into light waves. These light waves are transmitted over the fiber. Another converter placed near the sink converts the light waves back to electrical signals by photoelectric diodes. These electrical signals are amplified and sent to the receiver.

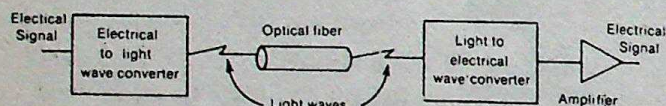


Figure 16.7. Illustrating transmission through optical fiber.

Some of the interesting features of optical fibers are :

(i) They provide high quality (low error rate) transmission of signals at very high speeds. Digital transmission speeds of 1 giga bits per second have been used with error less than  $1$  in  $10^9$  bits. Some transmissions that take about an hour over copper wires can actually be sent through optical fibers in under one second.

(ii) A normal fiber in a glass fiber cable is about the size of a human hair. In some large cities, there simply is no room for additional bulky copper wires. The conduits used to house the copper wires are typically of a fixed size, and replacing existing conduits for more copper wires would be too expensive. In this case, the use of smaller fiber optical data communications systems is very effective.

(iii) Optical fibers may be used to communicate either analog or digital signals. In analog transmission, the light intensity is varied continuously whereas in digital transmission the light source is turned on or off.

(iv) Fiber optic transmission are not affected by electromagnetic interference. Hence noise and distortion are also reduced with fiber optics.

(v) Optical fibers are most commonly used for point-to-point one way communications link because it is extremely difficult and expensive to tap a fiber optic cable at various points. This feature also provides security against unauthorised tampering of information.

While experts agree that the fiber optic transmission will be widely used in the future, there are still some technical difficulties to overcome. For instance, optical fibers do not currently allow complete switching or routing of light signals. As a result, at switching points, optical signals must be converted into electrical signals for routing and then back into light for further optical transmission. Despite such glitches, however, fiber-optic transmission systems are already on the market, and their popularity is growing rapidly.

## DIGITAL AND ANALOG TRANSMISSION

Data is propagated from one point to another by means of electrical signals which may be in digital or analog form. As shown in Figure 16.8(a), in an analog signal the transmitted power varies over a continuous range, for example sound, light and radio waves. The amplitude ( $v$ ) of analog signals is measured in volts and its frequency ( $f$ ) in hertz (Hz). The higher is the frequency of the signal, the more number of times it crosses the time axis. However, a digital signal is a sequence of voltage pulses represented in binary form (see Figure 16.8(b)).

Computer generated data is digital whereas the telephone lines used for data communication in computer networks are usually meant for carrying analog signals. When digital data are to be sent over an analog facility, the digital signals must be converted to analog form. The technique by which a digital signal is converted to its analog form is known as *modulation*. The reverse process, that is the conversion of analog signal to its digital form, at a destination device, is called *demodulation*.

Modulation is most often used for superimposing digital data on analog waves. There are three forms of modulation which are discussed below.



## PHASE MODULATION (PM)

Two binary values of digital data are represented by the shift in phase of the carrier signal. That is, a sine wave with phase =  $0^\circ$  represents a digital 1 and a sine wave with phase =  $180^\circ$  represents a digital 0. This technique is more noise resistant and efficient than both amplitude modulation and frequency modulation. For higher speed transmission of 9600 bits per second, phase modulation is more reliable and is preferred to other methods.

The process of modulation and demodulation, that is, the conversion of digital data to analog form and vice-versa, is carried out by a special device called a *modem* (modulator/demodulator). Hence, when an analog facility is used for data communication between two digital devices (say a terminal and a computer), two modems are required, one near each digital device. As shown in Figure 16.9, the digital signal generated at the terminal is converted to analog form by the modulator of the modem placed near it. The analog signal is transmitted through the telephone line which is converted to digital form by the demodulator of the modem placed near the computer. This digital data is processed by the computer. The processed digital information is modulated to analog form and returned via the telephone line to the terminal where the analog signals are demodulated to digital form for display on the terminal.

There are two main reasons for using modems. They allow higher speeds of transmission on a given analog line, and they reduce the effects of noise and distortion. Modems can do a lot more than simply transport data across telephone lines. A good modem can also perform tests and checks on how it is operating. Some modems are able to vary their transmission rate, commonly measured in baud. While the typical baud rate is 300, some modems can transmit at 1200 baud or even 9600 baud. Some of the more expensive modems actually contain microprocessors that allow them to operate and function under a large number of different circumstances. These modems are called *smart modems*.

A large number of portable terminals are now being used by salespersons, managers, engineers, and others to communicate with distant CPU's. A special type of modem called an *acoustic coupler* is used in these situations to provide the necessary interface. The acoustic coupler is attached (or built into) a portable terminal, and a standard telephone handset is then usually placed in rubber cups located on the coupler. The digital pulses produced by the terminal are converted into audible tones that are picked up by the handset receiver. The signals from these tones are sent to the CPU location where another modem converts them back to digital pulses.

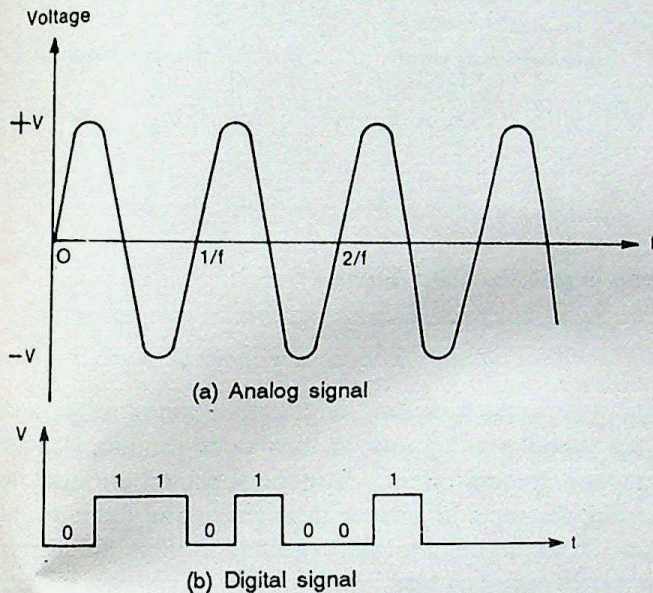


Figure 16.8. Analog and digital signals.

## AMPLITUDE MODULATION (AM)

Two binary values (0 and 1) of digital data are represented by two different amplitudes of the carrier signal keeping the frequency and phase constant. On voice grade lines, it is used upto 1200 bits per second. However, amplitude modulated signals are sensitive to impulse noises that arise due to electrical sparks near the transmission line.

## FREQUENCY MODULATION (FM)

Two binary values of digital data are represented by two different frequencies while the amplitude and phase are kept constant. Frequency modulation of digital signal is also known as *frequency shift keying* (FSK). It is less susceptible to error than amplitude modulation. Moreover, frequency modulation devices are easier to design because discrimination between two frequencies is simpler than detecting phase changes. Thus for medium speed communications (1200 to 2400 bits per second) the FSK scheme is preferred.



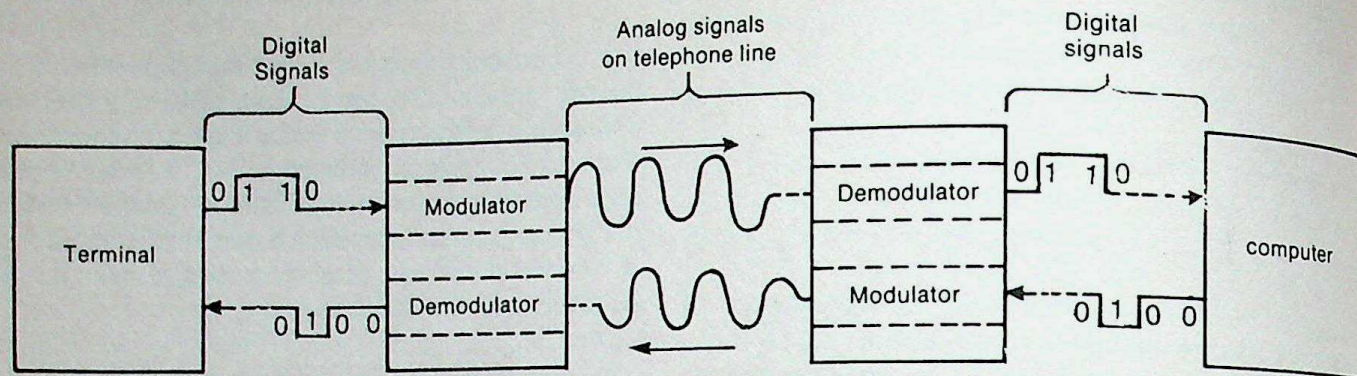


Figure 16.9. Illustrating the use of modems in data communications.

If data is sent over long distances, the analog signal can become weak and distorted. To avoid this, *amplifiers* are used. A communications amplifier is similar to the amplifier used in a home stereo system - it strengthens or amplifies a weak signal. Amplifiers are typically placed along communications lines between modems. Today, signals are also sent in digital mode. In this case modems are not needed. When digital mode of transmission is used over long distances, a *repeater* is used to receive and then transmit the signal. Like an amplifier, the purpose of a *repeater* is to strengthen a weak digital signal over long distances.

Digital transmission of digital data is more reliable than analog transmission of digital data. However, digital transmission facilities are currently available only to a limited extent. For example, digital signalling is impossible for satellite and whenever feasible the communication systems are converting to digital. The advantages of digital over analog for data transmission are :

1. Lower cost
2. No modems required
3. Higher transmission speeds possible
4. Lower error rate

It is expected that the years ahead will see ever-increasing digital transmission capabilities.

## COMMUNICATIONS PROCESSORS

In a data communications network, the task of network designers is to select and coordinate the network components so that the necessary data are moved to the

right place, at the right time, with a minimum of errors, and at the lowest possible cost. A number of communications processors (typical micro - or minicomputers) are used by network designers to achieve their goals. The functions of some of the commonly used communications processors are briefly described here.

## MULTIPLEXERS

There are many applications in which several terminals are connected to a computer. If each terminal is operating at 300 bits per second over a communications line (channel) that can operate at 9600 bits per second, then we see a very inefficient operation. It has been found that the capacity of a channel exceeds that required for a single signal. A channel is an expensive resource. Hence, for its optimal utilisation, the channel can be shared in such a way so as to simultaneously transmit multiple signals over it. The method of dividing a physical channel into many logical channels so that a number of independent signals may be simultaneously transmitted on it is known as multiplexing. The electronic device that performs this task is known as a multiplexer.

A multiplexer takes several data communications lines or signals and converts them into one data communications line or signal at the sending location. For example, as shown in Figure 16.10, there may be 4 terminals connected to a multiplexer. The multiplexer takes the signals from the 4 terminals and converts them into 1 large signal which can be transmitted over 1 communications line. Then, at the receiving location, a multiplexer takes the 1 large signal and breaks it into the



original 4 signals. Without multiplexers, you would have to have 4 separate communications lines.

Thus, with multiplexing it is possible for a single transmission medium to concurrently transmit data between several transmitters and receivers. There are two basic methods of multiplexing channels. They are frequency division multiplexing and time division multiplexing

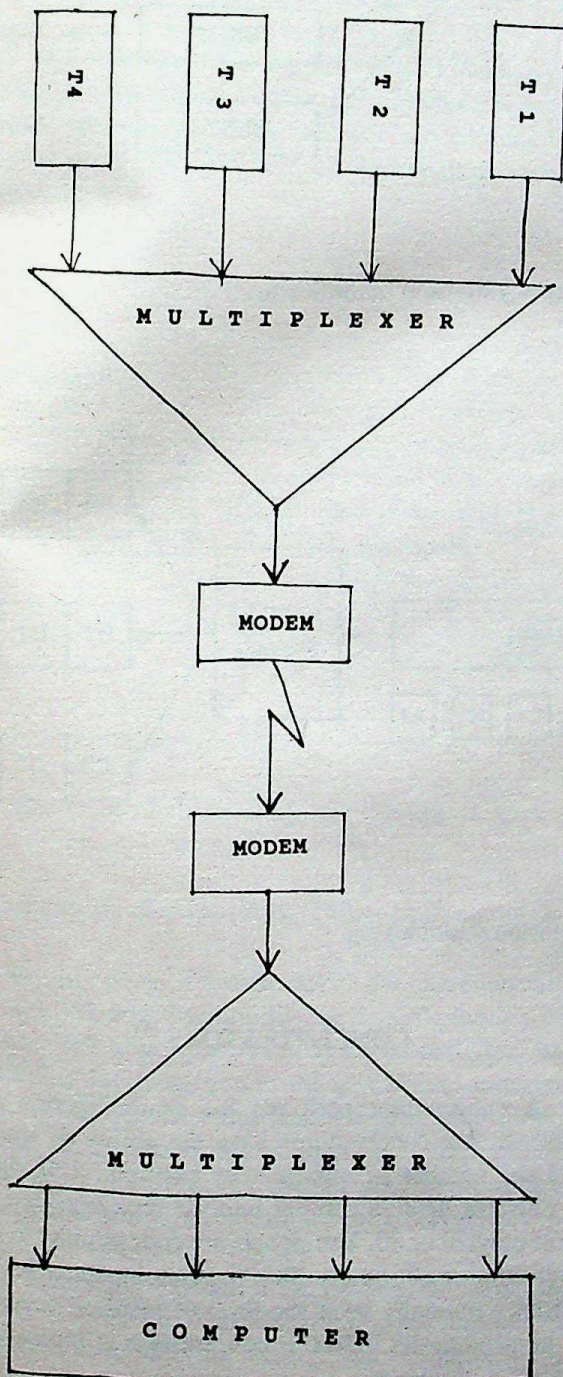


Figure 16.10. A multiplexed system.

1. *Frequency Division Multiplexing (FDM)*. The bandwidth or range of a medium exceeds that of any given signal. This fact is utilised for frequency division multiplexing. In FDM, the available bandwidth of a physical medium is split up (divided) into several smaller, disjoint logical bandwidths. Each of the component bandwidths is used as a separate communications line (channel). Figure 16.11 illustrates the process of FDM.

The best example of FDM is the way we receive various stations in a radio. Each radio station is assigned a frequency range within a bandwidth of radio frequencies. Several radio stations may be transmitting speech signals simultaneously over the physical channel which is "ether" in this case. A radio receiver's antenna receives signals transmitted by all the stations. Finally, the tuning dial in the radio is used to isolate the speech signal of the station tuned. In FDM, the signals to be transmitted must be analog signals. Thus, digital signals must be converted to analog form if they are to use FDM.

2. *Time Division Multiplexing (TDM)*. The bit rate of a transmission medium always exceeds the required rate of the digital signal. This fact is utilised for time division multiplexing. In TDM, the total time available in the channel is divided between several users and each user of the channel is allotted a time slice (a small time interval) during which he may transmit a message. The channel capacity is fully utilised in TDM by interleaving a number of data streams belonging to different users into one data stream. Streams of data sent through the physical channel must be demultiplexed at the receiving end. Individual chunks of message sent by each user is reassembled into a full message at the receiving end. The process of TDM is illustrated in Figure 16.12 for multiplexing three different signals. The same idea may be extended for multiplexing several signals.

TDM may be used to multiplex digital or analog signals. For communications of digital data, it is more convenient to transmit data directly in digital form and thus TDM is more appropriate. Moreover, communication between computers occurs in short fast bursts. Each burst would thus need the full channel bandwidth which is available to a signal in TDM. Besides this, TDM is generally more efficient as more subchannels can be derived because it is upto the network



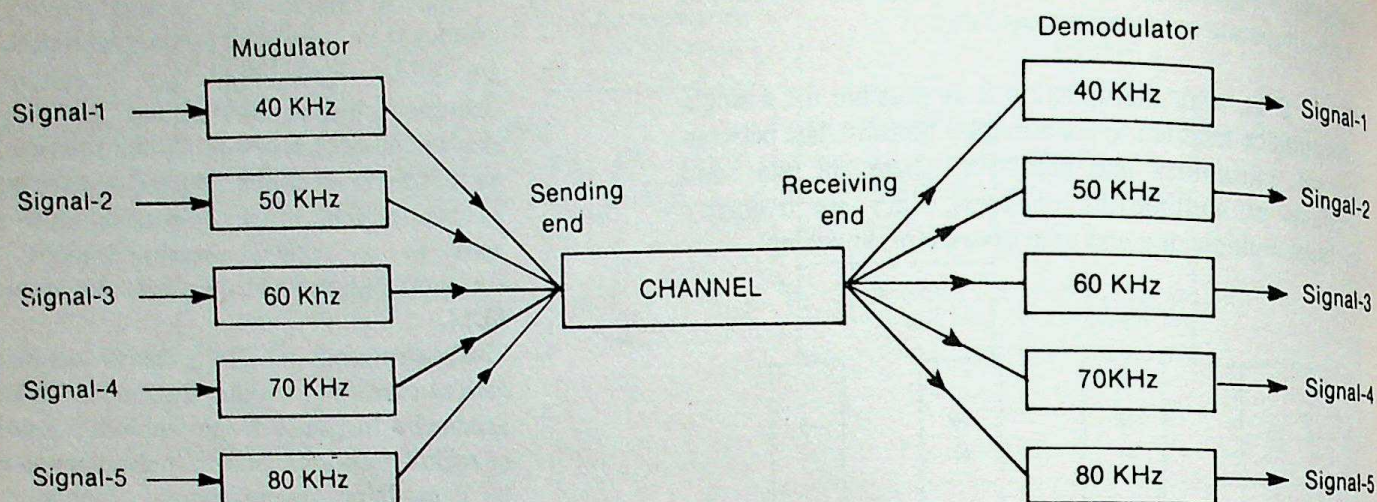


Figure 16.11. Illustrating frequency division multiplexing.

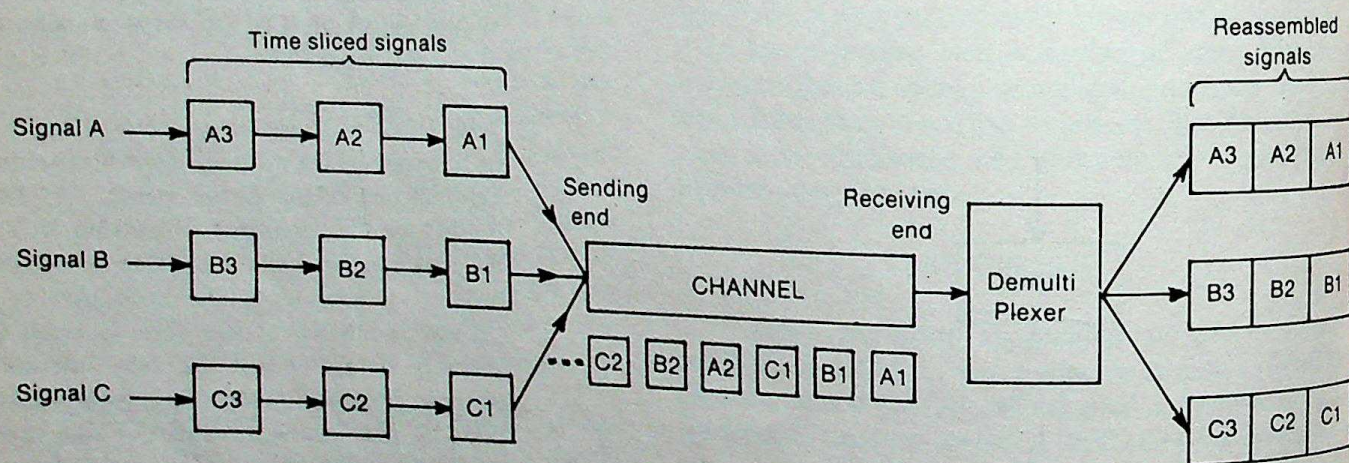


Figure 16.12. Illustrating time division multiplexing.

designers to allocate time slots to different channels. It is common to have 32 low speed terminals connected to one high speed line.

Whether or not to use multiplexing usually depends upon economics. The cost of high-speed modems and multiplexers is very high compared to the cost of low-speed modems. However, if line costs are high due to long distances, then multiplexing is cost effective. One serious disadvantage with multiplexing relates to a transmission line failure. If the line goes out, everything is dead. With individual lines only one terminal is likely to be lost.

## CONCENTRATORS

A concentrator performs the same function as a multiplexer, but concentrators have the ability to actually reduce the number of signals. For example, 100 signals from different devices coming into the concentrator could leave as only 70 or 80. The signals are concentrated to form a fewer number of signals. This requires intelligence. This intelligence normally takes the form of microprocessors or even minicomputers. Thus, a concentrator is basically an intelligent multiplexer.

The advantage of using a concentrator is that devices of varying speeds and types are connected to the



concentrator which in turn is connected to the host computer by high speed lines. Concentrators are especially useful where data communications cost are high, such as long-distance international communications. In general, concentrators are more expensive than multiplexers. For some long-distance and international data communications applications, however, the use of concentrators is cost justified.

### FRONT END PROCESSORS (FEP)

If information is to be moved from point to point in a computer network, then some type of control is necessary. Network processing (also known as communications processing) is the control of information movement between the

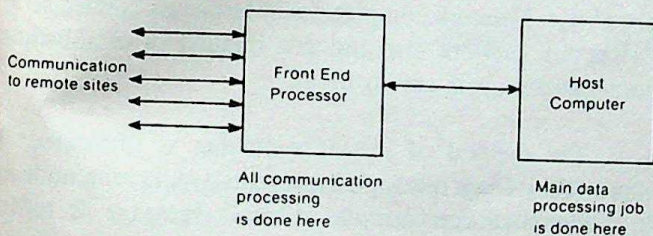


Figure 16.13. Illustrating the use of a front end processor.

various locations of the network. This is a substantial task.

In the early network days, the communications processing job was handled by the main computer. This began to place heavy additional loads on the CPU. Thus, the ability of the computer to do information processing, that is, the job for which it was intended, was seriously degraded. As a result, manufacturers introduced the front end processor. This is a separate CPU designed specifically to handle the network processing task (see Figure 16.13).

A front end processor is usually installed in the same location as the host computer. Its primary purpose is to off-load communications processing task from the host computer, thereby the host computer can be dedicated for applications and data processing jobs. It virtually acts like an intelligent terminal or like a secretary/receptionist to the main processor.

## ASYNCHRONOUS AND SYNCHRONOUS TRANSMISSION

Data transmission on a line is normally carried out in two different modes: asynchronous and synchronous. *Asynchronous* transmission is often referred to as *start-stop* transmission because of its nature, that is, the sender can send a character at any time convenient and the receiver will accept it. This is the characteristic of many terminals. When a terminal is connected to a computer and an operator manually presses keys on the terminal, the time spent between successive keystrokes would vary. Thus in asynchronous transmission, data is transmitted character by character at irregular intervals.

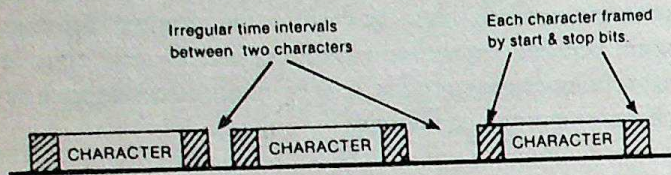
In order to enable the receiver to recognize a character when it arrives, the transmitter "frames" each character. Preceding the character is a *start bit* and following the character will be one or two (depending upon the system) *stop bits*. Thus for the 7-bit ASCII code, for each character the transmitter transmits the seven character bits, one parity bit, one start bit, and one or two stop bits for a total of either 10 or 11 bits. The concept of the character framed by start-stop bits is illustrated in Figure 16.14(a). Note that as each character is "framed" by a distinct start signal and a stop signal, the time taken between transmitting any two characters can be any length or none at all. Thus the computer might send a continuous stream of data to the terminal.

Asynchronous transmission is well suited to many keyboard type terminals and paper tape devices. The advantage of this method is that it does not require any local storage at the terminal or the computer as transmission takes place character by character. Hence it is cheaper to implement. The main disadvantage of asynchronous transmission is that the transmission line is idle during the time intervals between transmitting characters. If the lines are short, this is not bad because line cost would be low and idle time not expensive. Even though less efficient than synchronous transmission, it is also used with devices such as card readers and printers simply to reduce cost.

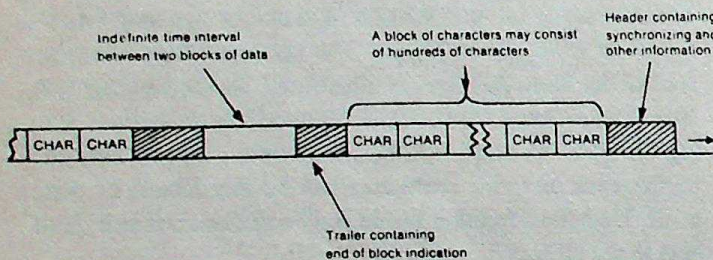
*Synchronous* mode of data transmission involves blocking a group of characters in somewhat the same way records are blocked on magnetic tape. Each block is then framed by header and trailer information. The header consists of synchronizing information which is used by the receiving device to set its clock in synchronism with the sending end clock. The header also contains information to identify sender and receiver. Following the header is a block of characters that contains the actual message to be transmitted (see Figure 16.14). The number of characters in



a block may be variable and may consist of hundreds of characters. The message characters in the block are terminated by a trailer. The trailer contains an end of message character followed by a check character to aid detection of any transmission error. Thus with synchronous transmission entire blocks of characters are framed and transmitted together.



(a) Asynchronous transmission



(b) Synchronous transmission

Figure 16.14. Modes of data transmission.

Synchronous transmission is well suited to remote communication between a computer and such devices as buffered card readers and printers. Here the logical data block length would be 80 or 132 characters. It is also used with buffered terminals and for computer to computer communication.

The primary advantage of synchronous transmission is its efficiency. Not only does it eliminate the need for individual start-stop bits on each character, but much higher data rates can be used than with asynchronous transmission. The period between blocks is kept small and the block itself is sent at nearly the maximum line speed. This ensures efficient utilisation of the transmission line. The main disadvantage is the need for local buffer storage at the two ends of the line to assemble blocks and also the need for accurately synchronized clocks at both ends. As a result synchronous equipment usually costs more.

## SWITCHING TECHNIQUES

In its simplest form, data communication takes place between two devices that are directly connected by some form of transmission medium - twisted wires, coaxial cables, microwave and satellite links. Often however, it is impractical or uneconomical for two devices to be directly connected. Instead, communication is achieved by transmitting data from source to destination through a network of intermediate nodes. These nodes provide a switching facility that moves data from node to node until the destination is reached. There are three different methods of establishing communication links between data sources and receivers in a communication network. These are: circuit switching, message switching and packet switching.

### CIRCUIT SWITCHING

It is the simplest method of data communication in which a dedicated physical path is established between the sending and the receiving stations through the nodes of the network. This method is used to connect two subscribers in a telephone. Network computers and terminals connected to a telephone network also use this method of establishing communication path among them.

The method of circuit switching is illustrated in Figure 16.15. Each rectangle represents a switching node of the communication network. When a message is to be communicated, a physical path is established between the sending station and the receiving station by physically connecting the incoming and outgoing line of each of the intermediate switching nodes that fall on the path. Once a circuit is established between the two stations, it is exclusively used by the two parties and the dedicated physical link between both ends continues to exist until the call is terminated either by the sender or the receiver. As soon as the connection is terminated by one of the two stations, the dedicated resources are deallocated and can now be used by other stations also.

Thus circuit switching involves three phases - circuit establishment, data transfer and circuit disconnection.

#### Advantages

1. The main advantage of circuit switching is that once the circuit is established, data is transmitted with no delay other than the propagation delay which is negligible (typically of the order of 6msec. per 1000 km).
2. It is suitable for low speed communication between



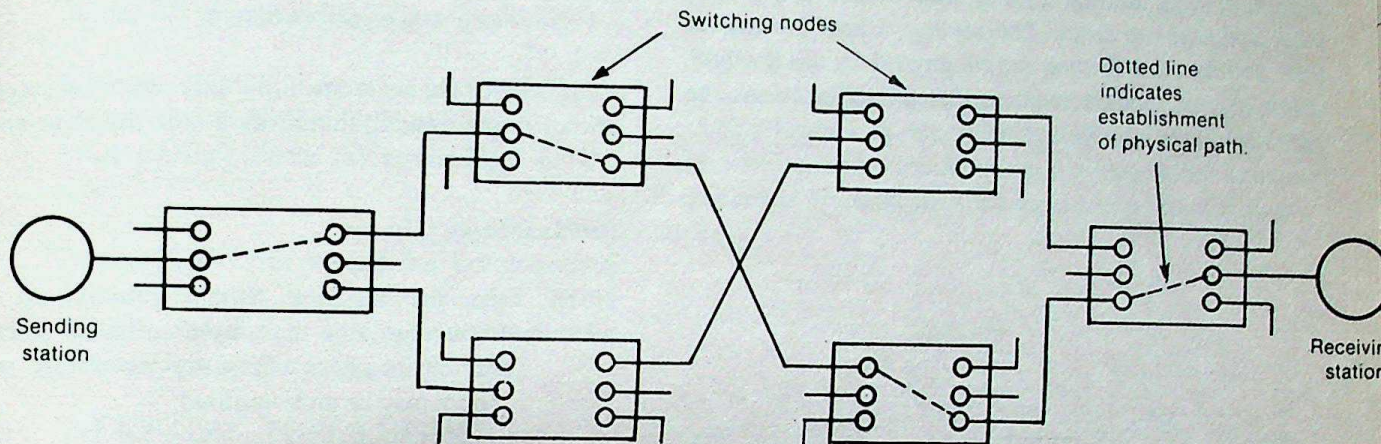


Figure 16.15. Illustrating the method of circuit switching.

a terminal and a computer because the method is simple and requires no special facilities.

computers that are connected using expensive high speed transmission lines.

3. Since a dedicated continuous transmission path is established, the method is suitable for long continuous transmission.

### Disadvantages

1. Prior to actual data transfer, the time required to establish a physical link between the two stations is of the order of 10 secs or more depending on the distance. For many computer applications e.g. point of sale verification, this elapse time is too large and unsuitable because communication among computers occur in bursts for short period with long silent period in between, during which the communication line is not used.
2. Since the physical link is a dedicated one, the channel capacity is dedicated to one source for the entire duration of a connection. Hence the network resources are not properly utilised.
3. The method proves to be very uneconomical when used for communication between

### MESSAGE SWITCHING

A message is a logical unit of information and can be of any length. In this method, if a station wishes to send a message to another station, it first appends the destination address to the message. After this, the message is transmitted from the source to the destination either by store-and-forward method or broadcast method.

As shown in Figure 16.16, in the store-and-forward method, the message is transmitted from the source node to an intermediate node. The intermediate node stores the complete message temporarily, inspects it for errors, and transmits the message to the next node based on an available free channel and its routing information. The actual path taken by the message to its destination is dynamic as the path is established as it travels along. When the message reaches a node, the channel on which it came is released for use by another message. In Figure 16.16, if a message is to be transmitted from station A to station B, it may take either path 1-2-3-4 or 1-5-4 depending on the availability of a free output path at that particular moment.



As shown in Figure 16.17, in the *broadcast* method, the message is broadcasted over a common medium known as broadcast channel. All the stations check the destination address of each message as they pass by and accept only those addressed to them. The routing delays inherent in store and forward method are eliminated in this method. However, this method requires that all stations must be connected to the broadcast channel.

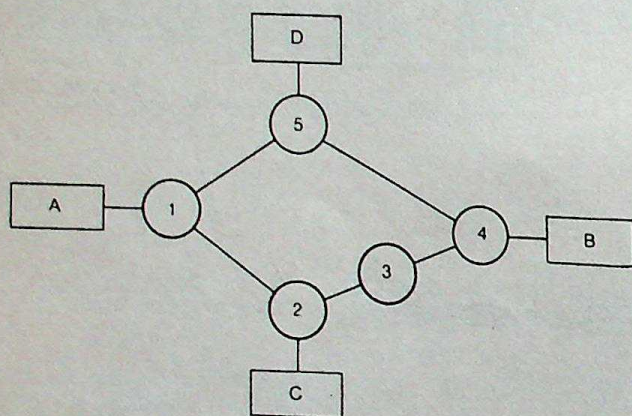


Figure 16.16. Illustrating store-and-forward method of message switching.

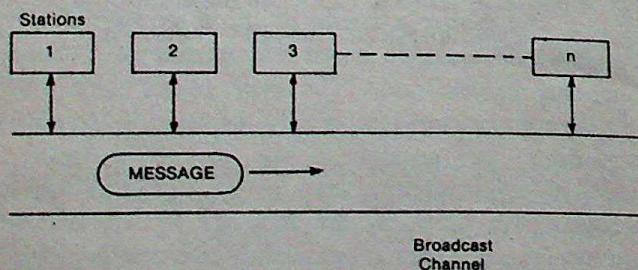


Figure 16.17. Illustrating broadcast method of message switching.

### Advantages

1. In message switching, no physical connection is required between the source and the destination as it is in circuit switching.
2. As channels are used only when messages are transmitted, this method uses the channels very effectively.

### Disadvantages

1. As the message length is unlimited, each switching node must have sufficient storage to buffer messages. In many cases this storage space may be under-utilised.
2. In message switching, a message is delayed at each node for the time required to receive the message plus a queuing delay waiting for an opportunity to retransmit the message to the next node.
3. The method is too slow for interactive/real time applications. Hence it is used when message delays are not critical.

## PACKET SWITCHING

This method works in a similar fashion as message switching. However, it overcomes the disadvantages of message switching technique because in this method routing is done on 'packet' basis and not on 'message' basis.

A message is split up into 'packets' of a fixed size (of the order of one or few thousand bits). Besides the block of data (part of a message) to be sent, a packet has a header that contains destination and source addresses, control information, message number, number of current and last packet, synchronisation bits, acknowledgement and error checking bytes, etc. Like message switching, the packets may be routed from the sender node to the destination node either by store-and-forward method or by broadcast method. In the store-and-forward method, the packets are stored temporarily at the intermediate nodes for error recovery purposes and is routed to next node based on an available free channel. The actual path taken by the packet to its destination is dynamic as the path is established as it travels along. Thus it is possible that different packets of the same message can be routed across different paths leading to the same destination depending upon line availability and congestion.



## Advantages

1. Unlike messages, packets are of small and fixed size. Hence the storage requirement for buffering the packets at intermediate nodes is minimal.
2. Since the routing is done on packet basis, the intermediate nodes do not have to wait for the entire message, hence the transmission is very fast.
3. The method is fast enough for interactive/real time applications. It is thus most suitable for "bursty" computer to computer communication and is widely used in long haul networks.

## NETWORK TOPOLOGIES

A network is a logical extension of a data communications system. In a computer network, two or more processors or computers are linked together with carriers and data communications devices for the purpose of communicating data and sharing resources. The term 'topology', in the context of a communications network, refers to the way in which the end points or stations of a network are linked together. It determines the data paths that may be used between any pair of stations of the network. Although the number of possible network configurations is seemingly limitless, there are three major network configurations. They are the star network, the ring network, and the completely connected network.

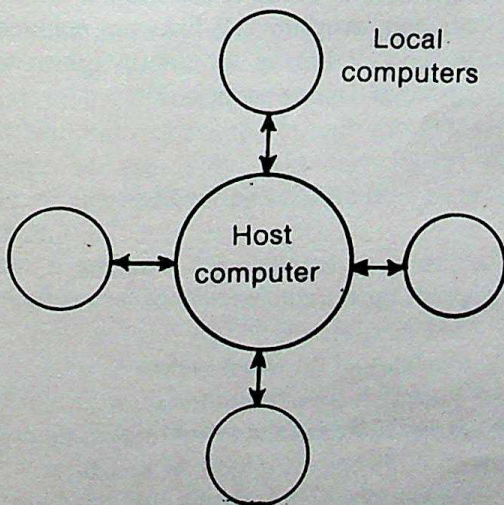


Figure 16.18. A star configuration of computer network.

## STAR NETWORK

In a star configuration of computer network, there is a host computer which is attached to local computers through multiple communication lines. As shown in Figure 16.18, the local computers are not linked directly to each other and can communicate only via the host computer. The routing function is performed by the host computer which centrally controls communication between any two local computers by establishing a logical path between them.

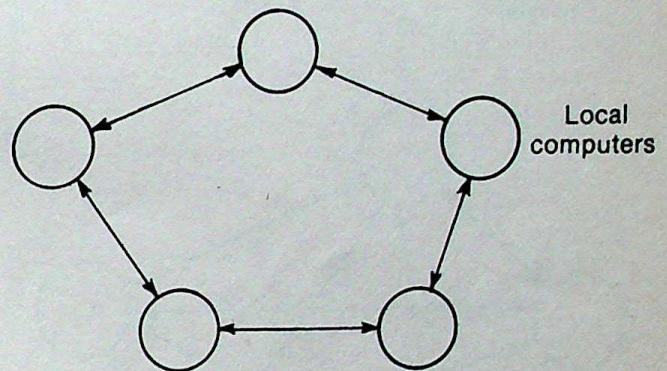


Figure 16.19. A ring configuration of computer network.

## Advantages

1. Star topology has minimal line cost because only  $n-1$  lines are required for connecting  $n$  nodes.
2. Transmission delays between two nodes do not increase by adding new nodes to the network because any two nodes may be connected via two links only.
3. If any of the local computers fail, the remaining portion of the network is unaffected.

## Disadvantage

1. The system crucially depends on the central node. If the host computer fails, the entire network fails.

## RING NETWORK

This configuration is a no-host or ring arrangement of communicating equals. That is, there is no main or controlling computer in the network. Figure 16.19



illustrates the circular or ring arrangement of a computer network. Each ring processor has communicating subordinates, but within the ring there is no master computer for controlling other computers.

Thus each node of a ring network must have simple communication capability. A node receives data from one of its two adjacent nodes. The only decision a node has to take is whether the data is for its use or not. If it is addressed to it, it utilises it. Otherwise it merely passes it on to the next node.

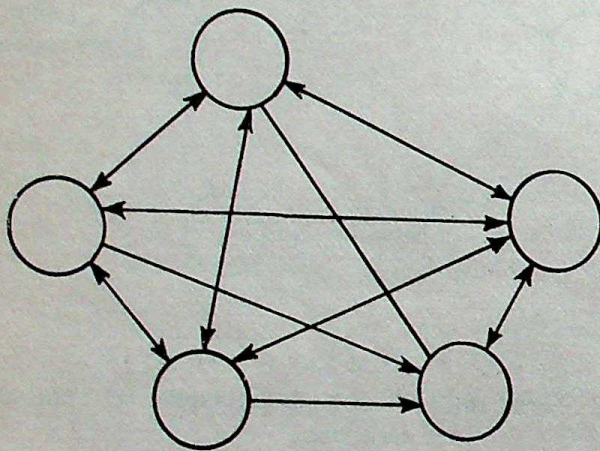


Figure 16.20. A completely connected computer network.

network increases the communication delays.

2. The ring network is not as popular as star network because of its more complicated control software.

## COMPLETELY CONNECTED NETWORK

As shown in Figure 16.20, a completely connected network has a separate physical link for connecting each node to any other node. Thus, each computer of such network has a direct dedicated link, called a point-to-point link with all other computers of the network. The control is distributed with each computer deciding its communication priorities.

### Advantages

1. This type of network is very reliable as any line breakdown will affect only communication between the connected computers.
2. Each node of the network need not have individual routing capability.
3. Communication is very fast between any two nodes.

### Disadvantage

1. It is the most expensive system from the point of view of line costs. If there are  $n$  nodes in the network, then  $n(n-1)/2$  links are required. Thus the cost of linking the system grows with the square of the number of nodes.

### Advantages

1. The ring network works well where there is no central-site computer system. It is a truly distributed data processing system.
2. It is more reliable than a star network because communication is not dependent on a single host computer. If one line between any two computers breaks down, or if one of the computers breaks down, alternate routing is possible.

### Disadvantages

1. In a ring network, communication delay is directly proportional to the number of nodes in the network. Hence addition of new nodes in the

## HYBRID NETWORK

The choice of topology for installing a computer network depends upon a combination of factors such as :

1. reliability of the entire system
2. expandability of the system
3. cost involved
4. availability of communication lines

5. delays involved in routing information from one node to another.



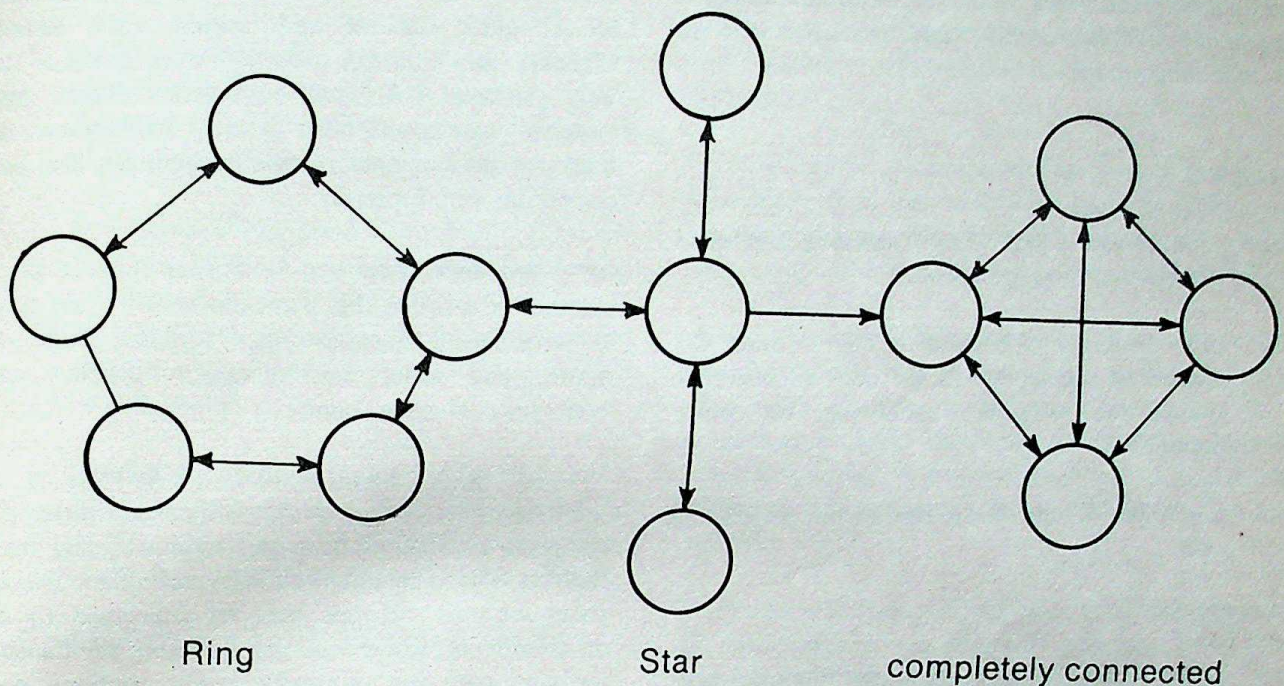


Figure 16.21.

A typical hybrid network having a combination of ring, star, and completely connected network configurations.

Different network configurations have their own advantages and limitations. Hence in reality, a pure star or ring or completely connected network is rarely used. Instead, an organisation will use some sort of hybrid network, which is simply a combination network. The exact shape or configuration of the network depends on the needs and the overall organisational structure of the company involved. In some cases, the hybrid network may have components of star, ring, and completely connected networks. A typical hybrid network of this type is shown in figure 16.21.

computer checks whether it is addressed to it. The message is picked up by the addressee computer which sends an acknowledgement to the source computer and frees the line. This method is known as 'multipoint' or 'multidrop' linkage of computers. It is also known as a 'broadcasting' network. This method is appropriate for use in

### MULTIPOINT NETWORK

There is yet another method of linking computers together in a network. In this method, a single transmission medium is shared by all nodes. As shown in Figure 16.22, all the computers are attached to the same line. When a particular computer wants to send a message to another computer, it appends the destination address to the message and checks whether the communication line is free. As soon as the line becomes free, it broadcasts (places) the message on the line. As the message travels on the line, each

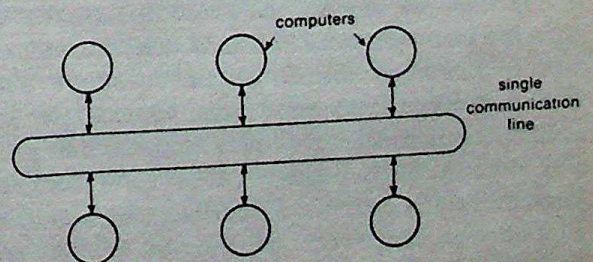


Figure 16.22 A multipoint or broadcasting network.



a local area network where a high speed communication channel is used and computers are confined to a small area. It is also appropriate when satellite communication is used as one satellite channel may be shared by many computers at a number of geographical locations.

### Advantages

1. The main advantage of a broadcasting network is the reduction in physical lines.
2. The reliability of network is high because the failure of a computer in the network does not effect the network functioning for other computers.
3. Addition of new computers to the network is easy.

### Disadvantages

1. Each computer connected to the line must have good communication and decision making capability.
2. If the communication line fails, the entire system breaks down.

## LOCAL AREA NETWORK AND WIDE AREA NETWORK

A *local area network* (LAN) is a digital communication system capable of interconnecting a large number of computers, terminals and other peripheral devices within a limited geographical area, typically under 1 Km across. LANs normally operate within a compact area such as an office building or a campus and is owned by the using organisation. The configuration of a LAN can be a star, a ring, or simply devices attached along a length of cable. In LAN, transmission channels generally use coaxial or fiber optic cables and special interface units rather than telephone lines and modems. Thus the transmission speed is very high. The attached computers may be of different types and be performing a variety of functions such as data processing, word processing, and electronic mail. The two main purposes of the local area network are to link computers within an organisation or campus so that they may share expensive peripheral devices, for example high speed printers or magnetic disks holding the data base, and to allow these computers to communicate with each other.

*Ethernet*, developed by Xerox Corporation, is a famous example of a LAN. This network uses a coaxial

cable for data transmission. Special integrated circuit chips called controllers are used to connect equipment to the cable, and small boxes called transceivers transmit and receive cable data at each station. Each station can exchange data with any other station or group of stations. This particular LAN can tie together Xerox computer systems, copiers, and other devices. Furthermore, devices made by other computer manufacturers can also be made compatible with Ethernet.

*Omninet*, developed by Corvus systems, is another example of a LAN. This particular network has the ability to interconnect microcomputer systems. Radio Shack, Apple, and other microcomputer systems can be interconnected using Omnet.

A *wide area network* (WAN) is a digital communication system which interconnects different sites, computer installations and user terminals, and may also enable LANs to communicate with each other. This type of communication network may be developed to operate nationwide or worldwide. In a WAN, the transmission medium used are normally public systems such as telephone lines, microwave and satellite links. Packet switching is most effective as public connection services are available for packet switched WANs. A WAN is also known as a *long haul network*.

The *ARPANET* of the Advanced Research Projects Agency of the U.S. Department of Defense is an example of WAN. This network connects about 40 universities and research institutions throughout the United States and Europe with about 50 computers ranging in size from minis to supercomputers. Another example of a WAN is the *INDONET* which is being planned by the Computer Maintenance Corporation, India. It links three IBM/4361 systems one each at Bombay, Calcutta and Madras and a PDP 11/44 system at Delhi. It is a network of large computer centres to give nationwide access and processing facilities.

### Differences Between LAN & WAN

1. A LAN is restricted to a limited geographical coverage of a few kilometers, but a WAN spans greater distances and may operate nationwide or even worldwide.
2. The cost to transmit data in a LAN is negligible since the transmission medium is usually owned by the user organisation. However, in case of a WAN, this cost may be very high because the transmission medium used are leased lines or



public systems such as telephone lines, microwave and satellite links.

3. In a LAN, the computers, terminals and peripheral devices are usually physically connected with wires and coaxial cables. Whereas, in a WAN there may not be a direct physical connection between various computers.
4. Data transmission speed is much higher in LAN than in a WAN. Typical transmission speeds in LANs are 0.1 to 100 mega bits per second. On the other hand, in a WAN the data transmission speed is normally of the order of 1800 to 9600 bits per second.
5. Fewer data transmission errors occur in case of a LAN as compared to a WAN. This is mainly because in case of a LAN, the distance covered by the data is negligible as compared to a WAN.

## COMMUNICATION PROTOCOLS

In a computer network, how are computers and related devices actually able to send information across data communications lines? The answer is data *communication software*. In fact, it is the data communication software that is responsible for holding all data communications systems together. It instructs computer systems and devices as to exactly how the data is to be transferred from one place to another. These procedures embedded in software are commonly called *protocols*.

Thus in data communication, a *protocol* is a set of rules and procedures established to control transmission between two points so that the receiver can properly interpret the bit stream transmitted by the sender. It provides a method for orderly and efficient exchange of data by establishing rules for the proper interpretation of controls and data transmitted as raw bits and bytes.

### ROLES OF PROTOCOL

In any computer network, data communications software normally performs the following functions for the efficient and error free transmission of data :

1. *Data sequencing*. It refers to breaking a long transmission into smaller blocks and maintaining control. That is, a long message is split up into smaller packets of fixed size. These packets are further fragmented into data frames. This technique is widely used in conjunction with error control techniques to reduce the amount of data that must be retransmitted in case of a detected error.

2. *Data routing*. Routing algorithms are designed to find the most efficient paths between sources and destinations. They can handle varying degrees of traffic on the present network configuration with optimal time utilisation. Normally, they are dynamic enough to accommodate network changes and growth.

3. *Flow control*. A communication protocol also prevents a fast sender from overwhelming a slow receiver. It ensures resource sharing and protection against congestion by regulating the flow of data on the communication lines.

4. *Error control*. Error detecting and recovering routines are also an important element of communications protocols. The most common method for correcting errors is to retransmit a block. This method requires coordination of the two stations that the block having error is discarded by the receiving station and is repeated by the transmitting station.

5. *Precedence and order of transmission*. There are well defined rules to condition all stations about when to transmit their data and when to receive data from other stations. It is ensured that all stations get a chance to use the communication lines and other resources of the network depending upon the priorities assigned to them.

6. *Connection establishment*. When two stations of a network want to communicate with each other, the communication protocol establishes and verifies a connection between the two.

7. *Data security*. Providing data security and privacy is also built into most communications software packages. It prevents access of data by unauthorized users because it is relatively easy to tap a data communications line.

8. *Log information*. Data communications software can also develop log information which consists of all jobs and data communications tasks that have taken place. Such information is normally used for financial purposes and the various users of the network are charged accordingly.

### THE OSI MODEL

The initial computer networks had their own set of standards and conventions that were quite hardware oriented. Each manufacturer used to develop their own communications protocols for their networks. For example, IBM launched SNA (Systems Network Architecture) in 1974. Similarly, DEC (Digital Equipment Corporation) launched its network in 1980 for use on the DEC range of computers. The architecture of DECNET (Digital Equipment Corporation Network) is known as DNA



(Digital Network Architecture). Like SNA, it is a private network architecture oriented only to DEC hardware. Other telecommunication protocols are BNA (Burroughs Network Architecture) by Burroughs Incorporation, DSN (Distributed Systems Network) by Hewlett Packard, PRIMENET (Prime Computers Network) by Prime Incorporation, etc. Thus the data communications protocols of one network were not compatible with any other network. Moreover, standards of the same network architecture also kept changing from time to time. For example earlier versions of SNA were not compatible with its subsequent versions. Such incompatibilities started creating bottleneck in the efficient and proper utilisation of network resources.

This problem was recognised by the International Standards Organisation (ISO) which established a sub-committee to develop an international standard on network architectures. The result was the Open System Interconnection (OSI) model, which is a framework for defining standards for linking heterogeneous computers in a packet switched network. Thus the standardized OSI protocols made it possible for any two heterogeneous computer systems, located anywhere in the world, to easily communicate with each other.

### CONCEPT OF LAYERING IN OSI MODEL

Most modern computer networks are designed in a modular or structured fashion for easy and efficient handling of the system. They are normally split up into a series of modules and are logically composed of a succession of layers or levels. Each layer offers certain services to the higher layers, shielding those layers from the details of how the offered services are actually implemented. Each layer has its own set of protocols. A particular layer of one machine communicates only with the corresponding layer of another machine using the protocols of this layer.

The OSI model is also designed in a highly structured way. As shown in Figure 16.23, it is a 7-layer architecture and defines a separate set of protocols for each layer. Thus each layer has a specific independent function which is explained in Figure 16.23.

1. *Physical layer.* The physical layer covers the physical interface between devices. It defines the electrical and mechanical aspects of interfacing to a physical medium for transmitting data. It also defines how physical links are set up, maintained and disconnected. In other words, it deals with the mechanical, electrical, procedural and functional characteristics of transmission. For example, in

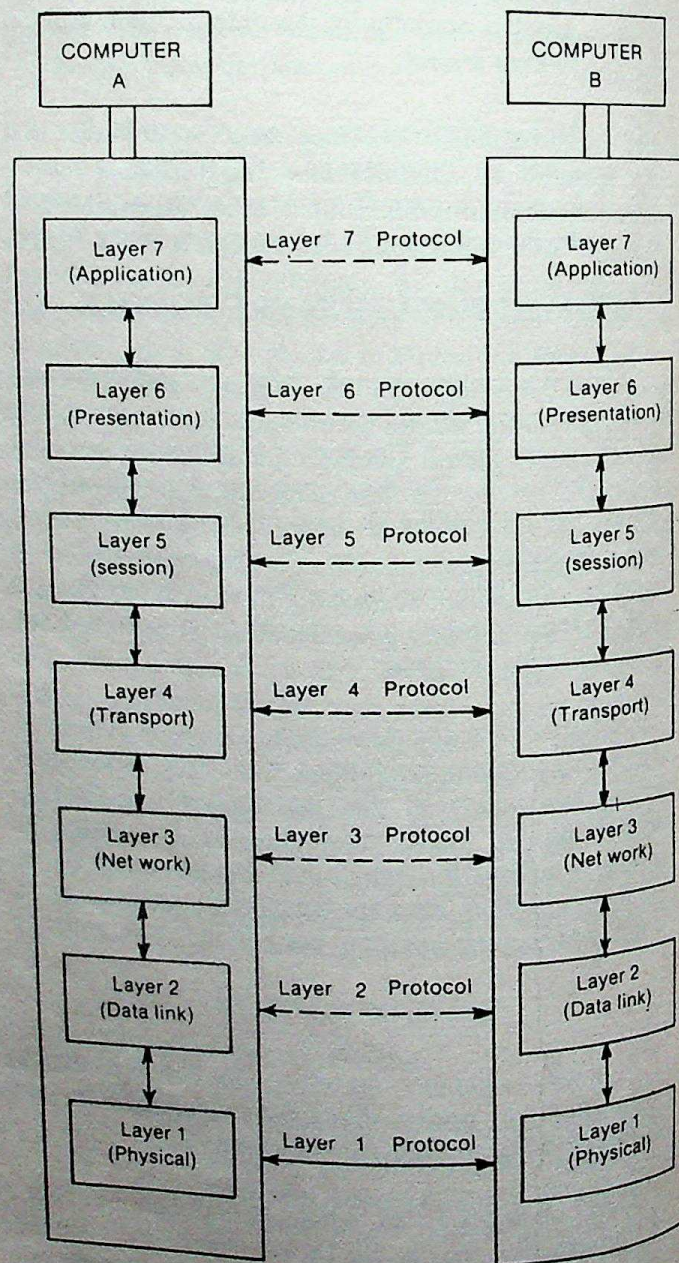


Figure 16.23. OSI network architecture showing layers, protocols and interfaces.



case of RS-232-C interface (used for interconnecting user terminals to computers), the mechanical aspect specifies a 25 pin connector to link the digital device to the modem; electrical characteristics relate to voltage levels and signal transmission rates; functional characteristics specify the functions of each of the pins and procedural characteristics specify the sequence of events for transmitting the data.

Thus, the physical layer transmits the data from the source side to the sink side as raw bits. In reality, data is transferred only between the lowest layers i.e. the physical layers of the source and destination machines. As shown in Figure 16.23, at the lowest (physical) layer, there is physical communication between the two machines as opposed to virtual communication at the higher layers. However, the higher layers are transparent to this.

**2. Data link layer.** The data link layer is responsible for establishing an error-free communications path between computers over the physical channel. It splits packets into data frames which are transmitted sequentially by the sender and the receiver transmits the acknowledgement. It performs checksum Cyclic Redundancy Check (CRC) and horizontal and vertical parity checks to detect and correct errors in the transmission of data. It also multiplexes different messages onto the same transmission channel for optimal line utilisation. It uses synchronous and asynchronous transmission techniques.

**3. Network layer.** The network layer sets up a logical path between the source and the destination computers of the network. The message to be transmitted is first fragmented into packets at this layer. Then it performs sequencing and error control of these packets. Thus the routing decisions are taken at the network layer.

**4. Transport layer.** Once a path is established between the sending and the receiving stations, the transport layer provides control standards for a communication session for enabling processes to exchange data reliably and sequentially.

The transport layer splits the message into smaller units and appends a header to each unit. The headers contain the sequence number of the message block. The transport layer also controls message flow between the sender and the receiver so that a fast sender does not overwhelm a slow receiver with data. It performs this by exchanging 'credit' messages with its peer layer. Moreover, in a multiuser, multiprogrammed environment, the transport layer multiplexes several message streams onto one physical channel.

**5. Session layer.** The session layer provides means of

establishing, maintaining and terminating a dialogue or a session between two end users. It specifies dialogue type - one way, two way alternate, or two way simultaneous and initiates a dialogue if the message is one of connection request i.e. a CALL. Further, it regulates the direction of message flow, i.e. it prevents an incoming message to be taken as an outgoing one.

**6. Presentation layer.** The presentation layer provides facilities to convert encoded transmitted data into displayable form for being displayed on a video terminal or printed on a printer. It may perform text compression, code conversion, or security encryption on the message.

**7. Application layer.** This is a user oriented layer which provides services that directly support the end user of the network. The offered services include file transfers, remote file access, data base management, etc. The application layer also provides network transparency to the end users so that they are ignorant of the physical distribution of the various network resources being utilized by them.

### AN EXAMPLE OF MESSAGE TRANSFER IN OSI MODEL

In order to illustrate the function of the various layers of OSI model, let us consider a simple example of message transfer from the sending station to the receiving station. As shown in Figure 16.24, let us assume that an end user, who is sitting at the sending station, sends a message M that has to be transmitted to the user at the receiving station. The message M invokes the application layer (7) which appends a header (H7) to the message and passes it on to the next layer (6) via the interface between layer 7 and layer 6. The presentation layer (6) performs text compression, code conversion, security encryption, etc. on the message. It appends a header (H6) to it and passes it on to the session layer (5). Depending upon the type of dialogue, the session layer establishes a dialogue between the sender and the receiver. It also regulates the direction of message flow. A header (H5) is appended to the message at this layer. The transport layer (4) now splits the message M into smaller units (M1 and M2) and appends a header (H4) to each unit. These headers contain the sequence number of the message block. The next layer, i.e., the network layer (3) further fragments the message blocks (M1 & M2) into packets and appends a header (H3) to each packet. Then the routing decisions are made for these packets. The network layer performs sequencing and error control of these packets and sets up a logical path between the source and the destination stations. The data link layer (2) once again splits the packets into data frames and attaches a header (H2) and a trailer (T2) to each of these data frames. Cyclic



### Disadvantages of a Distributed Data Processing System

1. There is lack of proper security controls for protecting the confidentiality and integrity of the user programs and data that are stored online and transmitted over network channels. It is relatively easy to tap a data communication line. One technique used to protect security and privacy over data communications lines is encryption. Basically, *encryption* is a coding device placed on either end of the data communications line, putting a very complex code on the data. This code is extremely difficult to break, although any code can be broken. At the other end of the data communications line, another encryption device is used to decode the signal into a meaningful message. However, encryption and other techniques are generally ineffective today in providing reliable security and privacy of confidential data and programs.
2. Due to lack of adequate computing/communications standards, it is not possible to link different equipments produced by different vendors into a smoothly functioning network. Thus, several good resources may not be available to the users of a network.
3. Due to decentralization of resources at remote sites, management from a central control point becomes very difficult. This normally results in increased complexity, poor documentation, and non-availability of skilled computer/communications specialists at the various sites for the proper maintenance of the system.
4. System reliability, flexibility, response time, vendor support, and the cost of telecommunications facilities used are some of the important factors that may disappoint network users in some cases.

### QUESTIONS

1. Why are computing and communications technologies merging? What are likely to be some of the effects of this?
2. Identify the basic elements of a communication system and the purpose of each.
3. Differentiate between simplex, half-duplex, and full-duplex modes of data transmission.

4. Which mode of data transmission is suitable for communication between a terminal and a computer?
5. "A full-duplex line is faster since it avoids the delay that occurs in a half-duplex circuit." Explain this statement.
6. Explain the terms "bandwidth" and "baud".
7. Why are the transmission mode and the bandwidth of communication channels of concern to an analyst designing a data communication system?
8. Differentiate between narrowband, voiceband, and broadband communication channels. Give a practical application of each.
9. What is a wire pair? In what situations are they suitable for use in data transmission?
10. What is a coaxial cable? Give some of its practical uses.
11. Explain how microwave systems can be used for communication between two distant stations.
12. How are communications satellites used? What are the possible advantages and limitations of using a communications satellite?
13. What is an optical fiber? How is it used for data communications? What are its advantages?
14. Differentiate between analog and digital transmission of data. Give their relative advantages and disadvantages.
15. What do you understand by modulation and demodulation?
16. Why is modulation used in signal transmission? Describe the three different forms of modulation used in signal transmission.
17. What are modems? What purpose do they serve in data communication systems?
18. What is an acoustic coupler?
19. What is a repeater?
20. When are public telephone lines used to connect



a terminal to a computer ? Are modems required for this purpose ? Why ?

21. What is a multiplexer ? Explain its function with the help of a diagram.
22. A company has a computer at its main office in Delhi linked to a terminal in each of its branch offices situated in the state capitals. It would like to use multiplexing for these terminals to reduce transmission costs. Is that possible ? Explain your answer.
23. Describe the two basic methods of multiplexing. Give use of both methods.
24. List out the differences between FDM and TDM. Which method is suitable for communication between computers and why ?
25. What is a concentrator ? Justify its use in a data communication system.
26. What is a FEP ? Illustrate its use .
27. Describe the asynchronous and synchronous modes of data transmission.
28. List out the relative advantages and disadvantages of asynchronous and synchronous modes of data transmission.
29. Explain how circuit switching method is used to link the sender and the receiver in a communication network. What are the advantages and disadvantages of this method of switching ?
30. Explain the store-and-forward method of message switching. Give the advantages and disadvantages of this message switching technique.
31. What is packet switching ? Why is this method used for digital data communication between computers ?
32. What is meant by network topology ? Describe three commonly used network topologies with their relative advantages and disadvantages.
33. What is a hybrid network ? Why are they used ?
34. Write a short note on multipoint network.
35. What is a LAN ? What are its main objectives ?
36. What is a WAN ?
37. Differentiate between a LAN and a WAN. Give one example of each.
38. What is a communication protocol ? What are the normal functions performed by these protocols ?
39. Why is layering used in the design of communication networks ?
40. Describe the layering concept in OSI model of network architecture with the functions of each layer.
41. What is meant by distributed data processing ?
42. What are the possible advantages and limitations of a distributed data processing system ?
43. What is encryption and why is it important for sensitive data ?







# GLOSSARY

The purpose of this glossary is to present definitions for some of the terms that are often used in the field of computers and data processing.

**Abacus** The earliest device that qualifies as a digital computer. It permits the user to represent numbers by the position of beads as a rack. Simple addition and subtraction can be carried out rapidly and efficiently by positioning the beads appropriately.

**Access time** The time interval between the instant at which data is called for from a storage device and the instant delivery begins.

**Accumulator** A local storage area, called a register, in which the result of an arithmetic or logic operation is formed.

**Acoustic coupler** A special type of modern (communications device) which allows an ordinary telephone to be used with a computer device for data transmission.

**Ada** A high level programming language named after Ada

**Augusta**, a friend of Charles Babbage. It is a general purpose programming language developed at the request of the U.S. Department of Defense for use in military applications.

**Adder** A logic circuit capable of forming the sum of two or more quantities.

**Address** An identification, represented in the form of a name, label, or number, for designating a particular location in storage area.

**Address register** A local storage register which contains the address of the next instruction to be executed.

**ALGOL (ALGOritmic Language)** An algebraic, high-level language similar to FORTRAN that is widely used in Europe.

**Algorithm** A sequence of precise and unambiguous instructions for solving a problem in a finite number of operations.

**Alphanumeric** Pertaining to a character set that contains



letters, digits, and usually other special characters such as the comma, dollar sign, plus sign, etc.

**ALU (Arithmetic Logic Unit)** The unit of a computing system which performs all mathematical and logical operations. It is one of the components of the central processing unit (CPU) of the computer.

**Analog computer** A computer that operates on data which is in the form of continuously variable physical quantities such as electrical current.

**ANSI (American National Standards Institute)** A U.S. based national organization that establishes uniform standards in several fields.

**APL (A Programming Language)** A very powerful high-level language that is well suited for specifying complex algorithms. It is a real-time language usually used in an interpretive and interactive manner and was developed primarily for scientific applications.

**Application program** Software designed for a specific purpose such as pay calculation, processing of examination results, stores accounting and inventory control, etc.

**Application programmer** A person who prepares application programs.

**Architecture** The organization and interconnection of the various components of a computer system.

**Artificial intelligence** A branch of computer science that deals with computers that possess reasoning, learning, and thinking capabilities that resemble those of humans.

**ASCII (American Standard Code for Information Interchange)** A standard coding system for computers. ASCII-7 is a 7-bit code and its extended version ASCII-8 is an 8-bit code.

**Assembler** A computer program which translates an assembly language program to its machine language equivalent.

**Assembly language** A low-level programming language in which mnemonics are used to code operations and alphanumeric symbols are used for addresses. This language lies between high-level languages (FORTRAN, COBOL, etc.) and machine language (the 1s and 0s the computer understands).

**Asynchronous communication** Communication between units operating independently.

**Audio response** An output medium that produces verbal responses from the computer system.

**Auxiliary storage** A storage that supplements the primary internal storage of a computer. Often referred to as secondary storage, this section of the computer's memory is characterised by low cost per bit stored, but it generally has an operating speed far slower than that of the primary storage.

**Background processing** The automatic execution of lower-priority (background) computer programs when higher-priority (foreground) programs are not using the system resources.

**Backup** Alternate facilities of programs, data files, hardware equipments, etc., that are used in case the original one is destroyed, lost, or fails to operate.

**Bandwidth** The range of frequencies available for data transmission. The wider the bandwidth of a communications system, the more data it can transmit in a given period of time.

**Base** The total number of digits (symbols) available to represent numbers in a positional number system.

**BASIC (Beginners All-Purpose Symbolic Instruction Code)** An easy-to-learn high-level interactive programming language frequently used with personal computers and in timesharing environments.

**Batch processing** The running of several computer programs one after another without the need of a human operator to run each program individually. This is also known as stacked job processing because several jobs are stacked together and processed in groups (batches) for efficient operation.

**Baud** A unit for measuring data transmission speed. It is used to describe the capacity of a carrier. In general usage, baud is identical to bits per second.

**BCD (Binary Coded Decimal)** One of the early coding systems used by computers which is based on the idea of converting each digit of a decimal number into its binary equivalent rather than converting the entire decimal value into a pure binary form. For example, the decimal number 42 is represented by 0100 0010 in 8-4-2-1 BCD notation.

**Binary** A characteristic or property involving a selection, choice, or condition in which there are two possibilities.

**Binary number system** A number system with a base of two.



It consists of two digits - 0 and 1.

**Bit** Acronym for binary digit which stands for one binary piece of information. This can be either 0 and 1.

**Block** A group of related items (records, characters, etc.) handled as a unit during input and output. A section of program coding treated as a unit.

**Blocking factor** The number of logical records in a physical record.

**Boolean algebra** An algebra that deals with logical propositions which are either true or false and to simplify such propositions. This algebra is suitable for use with binary number system and is very useful in designing logic circuits used by the processors of computer systems.

**Boolean function** A mathematical function in Boolean algebra. For example,  $w = x + y.z$ .

**Boolean variable** A variable used in Boolean algebra. It can assume a value true or false.

**Branch statement** An instruction that transfers program control to one or more possible paths.

**Broadband channel** The fastest carriers which have data transfer rates of 1 million baud (bits/second) or more. These data communication systems handle high volumes of data and are used for high speed computer to computer communication or for simultaneous transmission of data to several different devices.

**Buffer** A small storage area used to store information on a temporary basis for compensating the difference in rates of flow of data between various computer devices. For example, when data flows from an I/O device to the CPU, it passes through a buffer.

**Bug** An error in a computer program.

**Bus** Circuits that provide a communication path between two or more devices of a digital computer system. For example, the path between a CPU, storage, and peripherals.

**Byte** A fixed number of adjacent bits that represent a particular character or symbol. Normally a byte consists of eight bits.

**Cache memory** A small high speed memory which is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate.

**CAD (Computer Aided Design)** Use of computers to automate design operations.

**CAI (Computer Aided Instruction)** A general term that refers to a learning situation in which the student interacts with (and is guided by) a computer through a course of study aimed at achieving certain instructional goals. Also called CAE (Computer Aided Education).

**Call statement** A program statement which transfers program control to a subroutine.

**CAM (Computer Aided Manufacturing)** Use of computers to automate manufacturing operations.

**Canonical form** A Boolean function whose terms contain all variables (or their compliments). This is the unreduced form of the Boolean function in minterm or maxterm form.

**Card** A punched hole input-output media of constant size, thickness, and shape adapted for punching in a pattern which has meaning. Also called punched card.

**Card punch** An output device that places data onto computer cards. It converts data from a binary format in main storage to coded hole patterns in a punched card.

**Card reader** An input device that converts data coded onto punched cards into a binary format for entry into main storage. It transfers data contained on computer cards to the computer system.

**Card sorter** A unit record device not connected electronically to the computer system which is used to sort cards.

**Carrier** Any device that is used to transmit data from one location to another.

**Chain printer** A printer in which the characters are embossed on a chain or a band. The chain is in the form of loop which rotates at a high speed and print heads are activated to print specified characters.

**Channel (1)** A path for carrying signals between a source and a destination. **(2)** A track on a magnetic tape or a band on a magnetic drum.

**Character addressable storage** A storage device in which each character has one unique location with its own address. Another name for character addressable storage is variable word length storage.

**Character printer** A printer with a print mechanism that



prints one character at a time.

**Charge-coupled device (CCD)** A completely electronic storage device fabricated on semiconductor chips. It stores data as pockets of charge in a semiconductor.

**Chip** A thin wafer of silicon on which integrated electronic components are deposited.

**Circuit switching** The simplest method of data communication in which a dedicated physical path is established between the sending and the receiving stations through the nodes of the network for the complete duration of information exchange.

**COBOL (COMmon Business Oriented Language)** A high-level programming language developed for business data processing applications.

**CODASYL** An acronym for Conference On DATA SYstems Languages. This is a committee that helps to establish programming standards for various programming languages.

**Code** A set of rules outlining the way in which data may be represented within a computer system.

**Coding** The process of writing computer instructions in a programming language.

**Coding sheet** The form on which program instructions are written. Each programming language has its own unique coding sheets.

**Collate** To combine items from two or more sequenced files into a single one.

**Collating sequence** An ordering assigned to the characters of a character set to be used for sequencing purposes. A commonly encountered collating sequence involves the special characters, letters, and digits, in that order.

**COM (Computer Output Microfilm)** An output device that records computer output on microfilm.

**Combinational circuit** A group of logic gates interconnected to form a logic circuit.

**Comment** An entry in a computer program for the purpose of documentation or explanation. They are used within a program to assist anyone reading the source program listing. They do not contain any program logic and are ignored (not translated) by the language processor.

**Communications channel** A medium through which data (in the form of electrical signals) is transferred from one location to another.

**Communications processor** A processing unit that coordinates networks and data communications. Within a computer network, it ensures that data flows to and from different computer systems correctly and efficiently.

**Communications protocol** A set of rules and procedures established to interconnect and communicate between computers. It provides a method for orderly and efficient exchange of data by establishing rules for the proper interpretation of controls and data transmitted as raw bits and bytes.

**Communications satellite** A microwave relay station precisely positioned 36000 Kms. above the equator with an orbit speed that exactly matches the earth's rotation speed. It is used for data transmission between any two randomly chosen points in a very-very large area.

**Compile** To convert or translate a program written in a high-level language to an absolute or machine language form. Usually a single source statement yields more than one machine instruction.

**Compiler** A system software package that converts a high-level language program to machine language.

**Complement** For a number which has  $n$  digits in it, a complement is defined as the difference between the number and the base raised to the  $n^{\text{th}}$  power minus one. For example, complement of  $37_{10} = 10^2 - 1 - 37 = 62_{10}$ . In Boolean algebra, the complement of a variable is the reverse (NOT) of its value. For example, complement of  $A$  is  $\bar{A}$ .

**Computer** An electronic equipment designed to automatically accept and store input data, process them, and produce output results under the direction of a detailed step-by-step stored program of instructions.

**Computer graphics** The area of computer science which is concerned with the generation, manipulation and display of pictures with the aid of a computer.

**Computer network** A distributed data processing system in which multiple computers are linked together for the purpose of data communication and resource sharing.

**Computer operator** A person in the computer centre whose duties include setting up the processor and peripheral equipments, starting the program run, checking on



processor operation, and unloading equipments at the end of a run.

*Computer system* The various components (input and output devices, storage, CPU) of a computer integrated together to perform the steps called for in the program being executed.

*Connector symbol* Used in a flowchart to represent a junction in a flow line, this symbol is often used to transfer flow between pages of a lengthy chart.

*Console* The part of a computer system that enables human operators to communicate with the computer.

*Constant* A value, written into a program instruction, that does not change during the execution of the program.

*Control program* An operating system program which controls the operations and management of resources of a computer system. The control program's major functions are job scheduling, input/output scheduling, and program monitoring.

*Control Unit* The part of the central processor which directs the sequence of operations, interprets the coded instruction, and sees to the execution of program instructions.

*Cost/benefit analysis* A procedure for evaluation and selection of hardware and/or software in which lists are made of all the costs and benefits of each proposed data processing system.

*Counter* A device, register or storage location for storing integers that are suitably incremented or decremented to represent the number of occurrences of an event.

*CP/M (Control Program/Microprocessor)* A widely used disk operating system. It is a product of Digital Research Corporation and has become a standard for many 8-bit personal computers. Similarly CP/M-86 has become industry standard for many 16-bit personal computers.

*CPU (Central Processing Unit)* The control unit and the arithmetic logic unit of a computer system are jointly known as the CPU. It is the brain of any computer system. All calculations and comparisons are made inside the CPU and the CPU is also responsible for activating and controlling operations of other units of a computer system.

*CRT (Cathode Ray Tube)* An electronic tube with a TV like screen upon which information may be displayed.

*Cybernetics* The use of computers coupled with automatic

machinery to carry out complex operations.

*Cycle time* The time interval between the instant at which a read/write command is given to a memory and the instant when the next such instruction can be issued to the memory (also known as memory cycle time).

*Cylinder* In a disk pack, a set of corresponding tracks in all the surfaces is called a cylinder. All tracks of a cylinder are accessible by a single movement of the access mechanism.

*Daisy wheel printer* A letter-quality printer that uses a printing head with the appearance of daisy and/or a wheel. Each petal of the daisy wheel has a character embossed on it.

*DASD.* Acronym for Direct-Access Storage Device. See direct-access.

*Data* A collection of facts in raw form that become information after proper organization or processing.

*Data base* A collection of data files integrated and organised into a single comprehensive file system, which is arranged to minimize duplication of data and to provide convenient access to information within that system to satisfy a wide variety of user needs.

*Data base administrator* The one responsible for defining, updating, and controlling access to a data base.

*Data communications system* A system consisting of carriers and related devices used to transport data from one point to another.

*Data dictionary* The document that contains clear definitions of the data that will be used in setting up data base management systems.

*Data division* It is the third division of a COBOL program. Its purpose is to assign names and to allocate space within main memory for all files, records, and fields to be used by the program.

*Data element* A meaningful collection of related characters. Also called a field or data item.

*Data entry* The conversion of human readable data into a form a computer system can interpret. This is also called data preparation.

*Data processing* A series of operations that convert raw information (data) into useful information.



**Data processing system** A system that accomplishes data processing. It includes the necessary resources, which are people, materials, facilities, and equipments.

**Data processor** A digital device that processes data. It may be a computer, but in a larger sense it may gather, distribute, digest, analyze, and perform other organization or smoothing operations on data. These operations, then, are not necessarily computational. Data processor is a more inclusive term than computer.

**Data transfer rate** The speed at which data is transferred from main memory to another medium on which data are recorded. For magnetic tape, the data transfer rate is equal to the product of the tape speed and recording density.

**DBMS (Data Base Management System)** The software used for the management, maintenance, and retrieval of the data stored in a data base.

**Debugging** The process of finding and correcting program errors (bugs).

**Decimal number system** A number system with a base of 10. The ten allowable digits are 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. It is used in our day-to-day life.

**Decision symbol** A diamond-shaped symbol used in flowcharts to indicate a choice or branch in the processing path.

**Decision table** A table used for representing program logic. It displays the different conditions that could exist and the different actions that the computer should take as a result of these conditions.

**De Morgan's theorem** A theorem in Boolean algebra which states how to complement a Boolean expression.

**Design phase** A phase in the life-cycle of software system during which the detailed design of the system selected in the study phase occurs.

**Development phase** A phase in the life-cycle of a software system during which a system is constructed to meet the requirements specified in the design phase.

**Diagnostic routines** Programs used to print error messages by a computer to indicate system problems and improper program instructions.

**Digital Computer** A computer that works with discrete quantities. It uses numbers to simulate real-time processes. Compare with analog computer.

**Digitizer** An input device used to convert graphic and pictorial data into binary, numeric inputs for a digital computer.

**Direct-access** Pertaining to storage devices where the access time is effectively independent of the location of the data.

**Disk** A flat, circular plate coated with a magnetic material on which data can be stored by magnetization of portions of the flat surface.

**Disk operating system (DOS)** An operating system which contains the disk-oriented commands and that uses disk devices for permanent storage.

**Disk pack** A removable direct-access storage medium containing multiple magnetic disks mounted vertically on a single shaft.

**Diskette** A low-cost, thin, flexible magnetic disk storage device used on small computer systems. Also called a floppy disk.

**Distributed data processing (DDP)** The decentralization of a computer system through the use of multiple computers interconnected by a communications network. It facilitates data processing capabilities at the location of the end-user.

**Documentation** Documentation of a software system involves collecting, organizing, storing, and otherwise maintaining a complete historical record of programs and other documents used or prepared during the different phases of the system.

**Document reader** An optical input device that is used to read documents printed in a special type font.

**Downtime** The period during which a computer is malfunctioning or not operating correctly due to machine failures.

**Drum printer** A line printer that uses a solid, rotating, cylindrical drum on which the characters to be printed are embossed.

**Dumb terminal** A terminal that has no local processing capability.

**EBCDIC (Extended Binary Coded Decimal Interchange Code)** An 8-bit coding system developed by IBM that is used to represent characters in many modern computers.

**Edit To modify the form or format of the data by inserting**



or deleting characters where needed.

**Editor** A software used to interactively review and modify text materials and other program instructions.

**EDP (Electronic Data Processing)** Pertaining to data processing equipment that is predominantly electronic, such as an electronic digital computer.

**EDVAC (Electronic Discrete Variable Automatic Computer)** An electronic computing device similar to the ENIAC although smaller, faster, and having greater capability. It was built in 1952.

**Electronic funds transfer (EFT)** A general term referring to a cashless approach used to pay for goods and services. Electronic signals between computers are often used to adjust the accounts of the parties involved in a transaction.

**Electronic mail** A general term to describe the transmission of messages by the use of computing systems and telecommunications facilities.

**Electronic spreadsheet** An application package usually available with microcomputers that displays the equivalent of a work sheet made up of rows and columns. It may be used for anything that a person would compute or display in tabular form.

**Electrostatic printer** A high-speed printer that uses charged pins to form character matrices on chemically treated paper.

**Electrothermal printer** A high-speed printer that uses heated elements to create characters as matrices of small dots on heat-sensitive paper.

**Elementary data item** A data item which is not broken down into smaller units.

**Emulator** A program that permits one computer to execute the machine-language instructions of another computer of different make.

**End-of-tape (EOT) marker** A reflective marker that indicates the end of the usable tape.

**End user** Any individual who uses the information generated by a computer based system.

**ENIAC (Electronic Numerical Integrator And Calculator)** The first all-electronic digital computer developed by Mauchly and Eckert around 1946.

**Environment division** It is the second division of a COBOL

program. Its function is to describe the environment of the program, that is, those aspects that are dependent upon the physical characteristics of a particular computer.

**EPROM (Erasable Programmable Read Only Memory)** A semiconductor memory in case of which it is possible to erase information stored in it by exposing it to ultraviolet light. Later new information can be stored in it.

**Execution time** The total time required to execute a program on a particular system is called its execution time for that computer system.

**Executive routine** A master program in an operating system that controls the execution of other programs. It is also known as the executive, monitor, or supervisor.

**Facsimile (FAX)** Transmission of pictures, texts, maps, graphs, etc., over transmission lines, phone lines, and other carriers between geographically separated points. An image is scanned at a transmitting point and duplicated at a receiving point.

**Feasibility Study** A study to determine whether the proposed solution is technically and economically feasible in all respect.

**Fiber optic cable** A data transmission medium made of tiny threads of glass or plastic that is able to transmit huge amount of information at the speed of light.

**Field** In a record a meaningful collection of one or more related characters treated as a unit.

**Fifth generation computers** These computers will be introduced shortly. They will use a large number of processors working concurrently and independently. Simpler programming languages and knowledge based system implementations are expected in this generation.

**File** A collection of related records.

**Firmware** A sequence of instruction (software) that is substituted for hardware and stored in read-only memory (ROM).

**First generation computers** Computers built between 1949 and 1955 which used vacuum tubes and were programmed in assembly language. Few examples are ENIAC, EDVAC, EDSAC.

**First-in, first-out (FIFO)** A technique for processing jobs on a first-come, first-served basis.



**Fixed-head magnetic disk** A magnetic disk system that eliminates the use of an access mechanism by distributing all the read/write heads over the disk surfaces.

**Filp-flop** An electronic circuit which can be placed in one out of two stable states. Each state may be used to represent a binary digit.

**Floating-point numbers** Signed numbers held in a fraction-exponent format. For example, 3216 would be represented as  $0.3216 \times 10^4$  in floating-point notation.

**Floppy disk** See diskette.

**Flowchart** A pictorial representation that uses predefined symbols to describe either the logic of a computer program (program flowchart) or the data flow and processing steps of a system (system flowchart).

**Flowline** In a flowchart, flowlines with arrowheads are used to indicate the flow of operation, that is, the exact sequence in which the instructions are to be executed.

**Foreground processing** Automatic execution of high-priority (foreground) computer programs that have been designed to preempt the use of computer resources. Contrast with background processing.

**Format** The arrangement of input data, stored data, or output information.

**FORTAN (FORMula TRANslation)** A high-level, mathematically oriented programming language used for scientific and engineering applications.

**Fourth generation computers** Computers built between 1975 and now. They use large scale integrated circuits, semiconductor memories and powerful high-level languages and operating systems.

**Frequency division multiplexing** A method used to concurrently transmit data between several transmitters and receivers over a single transmission medium. The available bandwidth of a physical medium is divided into smaller, disjoint logical bandwidths and each of the component bandwidths is used as a separate communications line (channel).

**Front-end processor** A CPU designed specifically to handle the communications processing task. Its main purpose is to off-load communications processing task from the host computer, thereby the host computer can be dedicated for applications and data processing jobs.

**Full adder** An adder which adds three binary digits and outputs a result bit and a carry bit.

**Full duplex** A method of using a communication channel in which signals can be transmitted between a source and a destination in both directions simultaneously.

**General purpose computer** A computer capable of performing a variety of business and scientific applications.

**General purpose programming language** A programming language intended to solve a number of different types of problems.

**General software package** A software package, such as payroll and word processing, developed for a general market for many users.

**Generation** In computer talk, it is a step in technology. It provides a framework for the growth of the computer industry.

**Generator** A computer program that constructs other programs to perform a particular type of operation - e.g., a report program generator.

**GIGO (Garbage in, garbage out)** Pertains to the fact that most computer errors are not machine errors; they are data errors, caused by incorrect input data. Thus, incorrect input data results in inaccurate output.

**Graphic display terminal** A visual display terminal which has a screen to display a graph or drawing as well as alphanumeric information.

**Half adder** An adder which adds two binary digits and outputs a result bit and a carry bit (if any).

**Half duplex** A method of using a communication channel in which signals can be transmitted between a source and a destination in both directions, but only in one direction at a time.

**Hard copy** Printed or filmed output from a computer device in human readable form.

**Hardware** The physical components of a computer system such as electronic, magnetic, and mechanical devices.

**Hexadecimal number system** A number system using a base of 16. Its digits range from 0 to F. It is commonly used as a shortcut notation for groups of four binary digits.

**Hierarchical network** A communications network in which



computer or processors are connected in a tree-like structure.

*Hierarchical structure* A tree-like structure used to represent files and records in a data base system.

*High-level language* A programming language whose structure is application oriented and is independent of the structure of the computer. Each statement of such a language is translated into many machine language statements.

*HIPO Charts (Hierarchy plus Input-Process-Output charts)* It is both a design tool and an aid to program documentation which is concerned with what is done rather than how a particular activity is accomplished. It presents the software functions to be performed by a system in a top-down hierarchical sequence.

*Hollerith code* A particular type of 12-bit code that uses zone and/or digit area punches to represent alphanumeric data on 80-column punched cards.

*Host computer* The main control computer in a network of distributed processors and terminals.

*Hybrid computer* A combination of an analog and a digital computer. Such a computer system utilizes the measuring capability of an analog computer and the counting capability of a digital computer.

*Identification division* The first division of a COBOL program whose purpose is to identify the name of the program and any other information about the program that would be useful for identification purposes.

*Impact printer* A printer which prints characters by causing hammers to strike against the paper on which information is to be printed.

*Indexed file* A file that includes an index directory to facilitate random access.

*Information* The result of data processing which can be used to help people make decision.

*Ink jet printer* A printing device that uses a nozzle and sprays ink onto paper to form the appropriate characters.

*Input* The source data entered into a data processing system.

*Input device* A device used to bring information into a computer or other data processing devices; for example, a

card reader.

*Instruction* A command or order given to a computer. It normally consists of a code to indicate the operation to be performed and address(es) in memory where the operand(s) would be found.

*Instruction register* A register in CPU that holds the current instruction while it is being executed.

*Integrated circuit (IC)* Refers to the miniaturization of electronic circuits such that thousands of components are formed on a small chip of silicon or wafer. This chip is able to perform a variety of functions that in the past required several different electronic components.

*Intelligent terminal* A terminal having local processing capability. It has a built-in CPU and can perform specific functions such as editing data, controlling other terminals, etc.

*Interactive system* One that permits direct communication and dialog between system users and the operating program in the CPU.

*Inter block gap (IBG)* When several records are in one block, these gaps separate the blocks of records on magnetic tape.

*Interface* Electronic circuit used to interconnect I/O devices to a computer's CPU or memory.

*Internal storage* The addressable storage in a digital computer which is directly under the control of the CPU.

*Interpreter* A language processor that translates a statement of a high-level language and immediately executes it before translating the next source language statement. It is the most common language processor for BASIC.

*Inter record gap (IRG)* The separation or gap between records on a tape.

*I/O (Input/Output)* Pertaining to the techniques, media, and devices used for man-machine interaction.

*I/O bound jobs* Jobs that require more of I/O operations as compared to computational operations.

*IOCS (Input/Output Control System)* A set of routines for handling the many detailed aspects of input and output operations.

*IPO chart (Input-Process-Output chart)* An expansion of



a HIPO chart into additional levels of detail.

**ISAM (Index Sequential Access Method).** Relating to a file design technique whereby records organized in a sequential order can be accessed directly by use of an index (directory) based on some key or characteristic. Permits both sequential and random access of records.

**ISO protocol** A communication protocol to interconnect geographically dispersed heterogeneous computers. This protocol has been standardized by the International Standards Organisation (ISO).

**Job** A collection of specific tasks constituting a unit of work for a computer.

**Job Control Language (JCL)** A special purpose language used to describe to a computer's operating system the resource requirements of program fed to the computer.

**Jump** An instruction or signal which, conditionally or unconditionally, specifies the location of the next instruction and directs the computer to that instruction. A jump is used to alter the normal sequence control of the computer.

**Justification** Left justification (for alphabetic and alphanumeric data) means starting with the leftmost column. Right justification (for numeric data) means starting with the rightmost column.

**K (Kilo)** Used to represent  $2^{10} = 1024$  in computers.

**Key field.** A unique field in a record used to distinguish one record from another.

**Keypunch machine** A keyboard-actuated device that is used to record data on computer cards. It punches holes in a card to represent data.

**Key-to-disk** A device used to enter data onto a disk device.

**Key-to-tape** A device used to enter data onto a magnetic tape.

**Label** One or more characters used to identify a statement, an instruction, or a data field in a computer program.

**Label record** A machine-readable record that is used to identify a data file. It is the first record of the file.

**LAN (Local Area Network)** A digital communication system capable of interconnecting a large number of computers, terminals and other peripheral devices within a

limited geographical area, typically under 1 km across.

**Language processor** A software used to convert source program instructions to object or machine language instructions. Few examples are assembler, compiler and interpreter.

**Laser printer** A very high speed printer that uses a combination of laser-beam and electrophotographic techniques to create printed outputs at speeds in excess of 13,000 lines per minute.

**Latency time** In case of disk storage, the time taken for the desired record to come under the read/write head positioned over that track. Maximum latency time equals the time taken by disk to rotate once.

**Library routine** A tested routine maintained in a library of programs.

**Light pen** A pen shaped device that is used as an input device to computers by writing or sketching on the screen of a cathode ray tube.

**Line printer** A printer that appears to print one line at a time.

**LISP (LIST Processing)** A high-level programming language suitable for handling logical operations and non-numeric applications. It is used in the areas of pattern recognition, artificial intelligence and for simulation of games.

**Local storage** Storage areas, called registers, used by the CPU to interpret instructions and perform arithmetic and logical operations.

**Logical operators** Symbols used to show a logical relationship between two data items. Examples in FORTRAN are .EQ. for equal and .GT. for greater than.

**Logical error** An error that occurs when the actual logic of a program is different from the desired logic.

**Logic gate** An electronic circuit which operates on one or more input signals to produce standard output signals. For example, AND, OR and NOT gates. Logic gates are the building blocks of all the circuits in a computer.

**Loop** A sequence of instructions that is executed repeatedly until a terminal condition occurs.

**Low-level languages** Programming languages that normally translate from one source instruction to one object



instruction. These languages are machine dependent.

**LSI (Large Scale Integration)** The process of integrating a large number of electronic circuits on a single, small chip of silicon or other material.

**Machine language** A low-level language that is directly understandable by the computer system. Each model of a computer has a unique machine language.

**Macro flowchart** A flowchart that outlines the main segments of a program or that shows less detail.

**Macro instruction** An instruction in a source language that is equivalent to a specified sequence of machine instructions.

**Magnetic bubble memory** An electronic secondary storage made with solid-state electronic chips and have no moving parts. It uses the properties of certain materials, under applied magnetic fields, to represent binary 1s and 0s.

**Magnetic core** Tiny rings made of magnetic material that can be polarized to represent a binary 1 or 0.

**Magnetic disk** See disk.

**Magnetic-ink character recognition (MICR)** An input device that can read cards and paper documents printed with a special magnetic ink.

**Magnetic storage** Storage devices such as disks, drums, tapes, cores, etc. that utilize the magnetic properties of materials to store data.

**Magnetic tape** A secondary storage device that uses a long plastic strip coated with a magnetic material as a recording medium.

**Management information system (MIS)** An organized collection of people, procedures, and devices used to provide the right information to the right person at the right time for proper decision making.

**Mass storage** Storage systems that provide access to hundreds of billions of bytes of stored data. They are often referred to as archival storage because of the very large volumes of historical or backup data that they can store.

**Master file** A file containing relatively permanent data. This file is often updated by records in a transaction file.

**Maxterm** A Boolean quantity consisting of all terms (in its normal form or complement form) ORed together. Any

combination ( $2^n$  for  $n$  variables) of terms and complements is permissible, provided all are included in the term.

**Megabyte** One million ( $10^6$ ) bytes.

**Memory** A device or medium that can accept data, hold them, and deliver them on demand at a later time.

**Memory dump** A printout of the contents of memory.

**Merging** The combining of records from two or more ordered files into a single ordered file.

**Message switching** A method whereby messages to be transmitted between computers are all sent to a central computer, which gathers them and routes them to the appropriate destination(s).

**Microcomputer** The smallest category of computer fabricated using a microprocessor, and other integrated circuits, namely a ROM, RAM, and I/O interface chips.

**Microfiche** An output device that uses combined electronic, photo-optical, and electromechanical techniques to convert digital computer output to records that can be stored as rolls of microfilm or as frames of microfilm stored on cards called microfiche.

**Microfilm** See microfiche.

**Microflowchart** A flowchart with more details. It is also called a detailed flowchart.

**Microprocessor** A LSI chip which contains the entire CPU of a computer.

**Microprogram** Firmware programs residing in read-only memory (ROM) which aid the control unit in directing operations of a computer system.

**Microsecond** One-millionth of a second.

**Millisecond** One-thousandth of a second.

**Minicomputer** A relatively fast but small and inexpensive computer with somewhat limited input/output capabilities.

**Minterm** A Boolean quantity consisting of all terms (in its normal form or complement form) ANDed together. Any combination ( $2^n$  for  $n$  variables) of terms and complements is permissible, provided all are included in the term.

**Mnemonic** Any kind of mental trick we use to help us remember. For example, a computer may be designed to



**interpret the machine code of 1111 (binary) as the subtract operation, but it is easier for a human being to remember it as SUB.**

**Modem (Modulator-demodulator)** Devices used to convert digital signals (to be communicated over an analog channel such as telephone line) to sine waves at the sending end and back to digital signals at the receiving end.

**Modular approach** Dividing a project into segments and smaller units in order to simplify the analysis, design, and programming efforts.

**Modulation** The technique by which a digital signal is converted to its analog form for transmission over an analog facility.

**Monitor routine** See executive routine.

**MSI (Medium Scale Integration)** A circuit with about 100 transistors fabricated on a single chip.

**Multiplexing** The method of dividing a physical channel into many logical channels so that a number of independent signals may be simultaneously transmitted on it.

**Multiprocessing** A term used to describe interconnected computer configurations or computers with two or more independent CPUs that have the ability to simultaneously execute several programs.

**Multiprocessor** A computer system consisting of two more CPUs under a common control.

**Multiprogramming** The name given to the interleaved execution of two or more different and independent programs by the same computer.

**Nanosecond** One-billionth ( $10^{-9}$ ) of a second.

**Narrowband channel** Communication channels that handle low volumes of data, typically from 45 to 300 baud. They are used mainly for telegraph lines and low speed terminals.

**Network** An interconnection of computer systems and/or peripheral devices with carriers and data communications devices for the purpose of exchanging data and information.

**Network topology** The structure of interconnection of nodes of a computer network.

**Node** An end point of a branch in a network, or a common junction of two or more network branches.

**Nonimpact printer** A printer which performs some type of operation to the paper instead of physically striking it (as in the case of an impact printer). To print characters with nonimpact printers, the paper can be sprayed with ink, magnetized, electrically charged, heated, placed under pressure, or struck by laser beams.

**Nonvolatile storage** A storage medium that retains its contents even in the absence of power.

**Object program:** A fully compiled or assembled program that is ready to be loaded into the computer. It results from the translation of a source program by a language processor.

**Octal number system** A number system with a base of 8. The octal digits range from 0 to 7. It is commonly used as a shortcut notation for groups of three binary digits.

**Offline** A device or system not directly connected to the CPU.

**Online** A device or system directly connected to the CPU.

**Operand** The part of a machine level instruction which tells the central processor the location of the data to be manipulated.

**Operating system** An integrated set of programs that is used to manage the various resources and overall operations of a computer system.

**Operation code (op code)** The part of a machine level instruction which tells the central processor what has to be done.

**Operation phase** The life-cycle phase during which the system constructed in the development phase is used.

**Optical bar-code reader** An input device that is able to interpret combinations of marks (bars) that represent data.

**Optical Character Reader (OCR)** An input device which can read characters directly from an ordinary piece of paper by use of a scanning mechanism. These characters are written in special type fonts.

**Optical Mark Reader (OMR)** An input device that is able to interpret pencil marks on paper media.

**Output** The finished result of processing by a system.

**Output unit** The unit of a computer system that supplies information and results of computation to the outside world.



example, one program can undergo input, another program can be processed, and a third program can undergo output all at the same time.

**Packed decimal** A modified form of zoned decimal that places two decimal digits into each byte of the field with the exception of the rightmost byte, which contains one digit and the sign of the field.

**Packet switching** A method of communication between computers in a network in which blocks of messages to be transmitted are formed into packets and placed on the channel. Each packet contains source and destination addresses, synchronizing, error correction and control bits. The packets are routed using the source and destination addresses.

**Page printer** A high-speed printer with a mechanism that appears to print an entire page at one time.

**Page reader** A high-capacity optical input device that is able to scan and interpret an entire page that is typed in a special font.

**Paper tape punch** An output device that converts data from a binary format in main storage to coded hole patterns punched into paper tape.

**Paper tape reader** An input device that converts data punched into paper tape into a binary format for entry into main storage.

**Paralled adder** An adder in which all the bits of the two operands are added simultaneously.

**Parallel operation** A system changeover method whereby data is processed by both the old and the new system until performance of the new system is verified satisfactorily.

**Parity bit** An extra bit added to a string of bits that enables the computer to detect internal errors in the transmission of binary data.

**Pascal** A high-level programming language named after Blaise Pascal that facilitates the use of structured programming techniques.

**Password** A code by which a user gains access to a computer system. It is used for security purposes.

**Peripherals** The various input/output devices and auxiliary storage units of a computer system.

**Personal computer** A small and inexpensive computer

(usually a microcomputer) used by individuals for carrying out personal jobs or for applications such as entertainment, home management, and hobbies.

**PERT** An acronym for Program Evaluation and Review Technique, a technique used to facilitate the implementation of a computer system.

**Phase modulation** A form of modulation in which two binary values of digital data are represented by the shift in phase of the carrier signal. That is, a sine wave with phase =  $0^0$  represents a digital 1 and a sine wave with phase =  $180^0$  represents a digital 0.

**Phased replacement** A system changeover method in which the complete changeover to the new system takes place incrementally over a period of time. The new system is gradually implemented part by part and the old system is gradually phased out.

**Picosecond** One trillionth of a second.

**Pixel** A picture element. It is used to represent one point in a raster scan display device.

**PL/I (Programming Language One)** A high level programming language designed for handling both scientific and business applications.

**Plotter** An output device that converts computer output into a graphic, hardcopy form.

**Pointer** A data item in one record that contains the location address of another logically related record.

**Point-of-sale (POS) device** An I/O device capable of immediately updating sales and inventory records at a central CPU and producing a printed sales transaction receipt.

**Primary memory** A section of the CPU that holds program instructions, input data, intermediate results, and the output information produced during processing. Also known as primary storage, internal storage and main memory.

**Printer** An output device used to produce hard copy of computer output that is readable by humans.

**Problem-oriented languages** High-level programming languages designed for the convenient expression of a given class of problems; for example, RPG.

**Procedure division** The last division of a COBOL program that contains the logic of the program, i.e., all the



given class of problems; for example, RPG.

**Procedure division** The last division of a COBOL program that contains the logic of the program, i.e., all the instructions that the computer is to follow in solving our problem.

**Procedure-oriented languages** High-level programming languages designed for the convenient expression of procedures used in the solution of a wide class of problems; for example, FORTRAN and COBOL.

**Process bound jobs** Jobs that require more of computational operations (CPU time) as compared to I/O operations (I/O time).

**Processing** Performing arithmetic operations and/or logically manipulating input data in order to convert them into a desired output.

**Processing symbol** A rectangular figure used in flowcharts to indicate a processing operation.

**Process loop** See loop.

**Processor** A unit of a computer system that interprets instructions and executes them.

**Program** A set of sequenced instructions used to direct and control the operations of the computer in order to solve a problem or to perform a particular task. Also known as a routine.

**Program counter (PC)** A register in CPU which is used to store the address of the next instruction to be executed.

**Program Design Language (PDL)** See pseudocode.

**Program library** A collection of complete programs, subroutines, and program modules that have already been developed, tested, and documented usually as a result of other programming projects.

**Programmer** One who designs, writes, tests, and maintains computer programs.

**Programming language** A language used to express algorithms in computer understandable form.

**PROM (Programming Read-Only Memory)** Similar to read only memory with the exception that these chips can be reprogrammed by using special external equipments.

**Protocol** See communications protocol.

**Pseudocode** A programming analysis tool that is used for planning program logic. It is an imitation of actual computer instructions written in ordinary natural language such as English.

**Radix** Same as base. See base.

**Random access memory (RAM)** A storage device in which the time to retrieve stored information is independent of the address where it is stored.

**Real-time systems** Pertains to online computer processing systems which receive and process data quickly enough to produce output to control, direct, or affect the outcome of an ongoing activity or process.

**Record** A collection of related items of data treated as a unit.

**Record length** A measure of the size of a record, usually specified in units such as characters.

**Register** See local storage.

**Remote access** Accessing a computer from a distant station using communication facilities.

**Remote job entry (RJE)** A particular type of batch processing, where the input and output devices are located away from the central computer facility. The equipments for a RJE station can include terminals, card readers, card punching devices, printers, etc.

**Report program generator (RPG)** A business oriented, general purpose programming language designed to generate the output reports resulting from the processing of common business applications.

**Response time** The total elapsed time between submission of command and data to a computer and getting the result of computation.

**Ring network** A computer network in which there is no host computer for controlling other computers and in which all stations are equal.

**Robot** An automatic machine that performs routine, seemingly human tasks.

**ROM (Read Only Memory)** Special memory chips containing instructions which can be read only, therefore preventing accidental destruction of the instructions. ROM is used to store firmware.



*Routine* See program.

*RS-232-C interface* A standard interface used for interconnecting user terminals to computers. It was defined by Electronics Industrial Association (USA).

*Run time* The time required to complete a single, continuous, execution of an object program.

*Secondary storage* See auxiliary storage.

*Second generation computers* Computers built during the period 1955-64 which used transistors in CPU, magnetic core main memories, and high-level languages like FORTRAN and COBOL for programming.

*Seek time* In a disk system, the time required for a read/write head to move to the track where the record to be read or written is stored. Maximum seek time equals the time taken for the head to move from the outermost track to the innermost track.

*Semiconductor storage* A storage device whose storage elements are formed as solid-state electronic components on an integrated circuit chip.

*Sequential processing* A technique in which a number of similar items or transactions to be processed are grouped together and processed in a designated sequence.

*Serial access* A storage device or medium where the access time is dependent upon the location of the data. Magnetic tape is a typical serial access medium.

*Serial adder* An adder in which the bits of the operands are added one after another.

*Service programs* Programs commonly used in all computer centres for providing better service to users. They include programs to prepare object programs for execution, to store programs on a magnetic disk, sort data stored on secondary storage devices, etc. They are part of the operating system.

*Simplex* A way in which data can be transmitted so that the data can only flow in one direction.

*Simulation* To represent and analyze properties or behavior of a physical or hypothetical system by the behavior of a system model.

*SNOBOL* An acronym for string oriented symbolic language. A high-level language designed to manipulate strings of characters and used for nonnumeric applications.

*Soft copy* Computer output which is displayed on the screen of a terminal and provides no permanent copy.

*Software* The set of computer programs, procedures, and associated documentation related to the effective operation of a computer system.

*Solid state* Electronic components whose operation depends on the control of electric or magnetic phenomena in solids, such as transistors and diodes.

*Sort* The process of arranging data into a desired sequence.

*Source document* A document on which data that are to be recorded in machine-readable code originates, that is, the original hand written or typewritten document or a time card

*Source program* A program written in a symbolic or high-level source language such as assembly language, COBOL, BASIC, etc.

*Special character* A graphic character that is neither a letter, a digit, nor a space character; for example, the dollar sign, comma, period, etc.

*Special purpose computer* A computer that is meant for doing only a particular type of jobs.

*Special purpose programming language* A programming language designed to handle one specific type of problem or application.

*Speech recognition* The ability to input data directly into a computer system by speaking to it.

*Spooling* A technique that has been successfully used on a number of computer systems to reduce the speed mismatch between slow speed I/O devices and fast CPU. It is the process of placing all data that comes from an input device or goes to an output device on either a magnetic disk or tape which are faster devices as compared to a card reader or a line printer.

*SSI (Small Scale Integration)* An electronic circuit with about 20 transistors fabricated on a silicon chip.

*Stack* A memory in which information which is stored last is on top and is retrieved first. Also known as LIFO (Last-in-first-out) storage.

*Stand-alone* In a distributed data processing system, it refers to a computer system that has an independent (from a central-site computer) processing and storage capability.



**Stand-alone** In a distributed data processing system, it refers to a computer system that has an independent (from a central-site computer) processing and storage capability.

**Star network** A network having a central host computer system that is attached to local computers through multiple communication lines. The local computers are not linked directly to each other and can communicate only via the host computer.

**Statement** In programming, an expression or generalized instruction in a source language.

**Storage** See memory.

**Store and forward method** A method of message transmission in a computer network in which messages are sent to a central computer which receives them, stores them and forwards them to the specified destinations.

**Stored program computer** A computer where the program to solve a problem and the necessary data are stored in its memory.

**Structured design** A system design approach in which a difficult problem is broken into smaller problems that are small enough to manage but independent enough to solve separately.

**Structured programming** An organized approach to programming involving the use of three basic control structures - sequence, branch, and loop and the use of top-down concepts to decompose main functions into lower-level components for modular coding purposes. This technique is concerned with improving the programming process through better organization of programs and better programming notation to facilitate, correct, and clear descriptions of data in control structures.

**Study phase** A system life-cycle phase during which a problem is identified, alternative solutions are studied, and the most feasible solution is selected.

**Subroutine** A standardized program written in such a way that it can be used as part of another program whenever necessary. A subroutine is normally invoked through other programs by the use of CALL statements.

**Subsystem** A system that is part of a larger system.

**Supercomputer** Computer systems characterized by their very large size and very high processing speeds. They are generally used for complex scientific applications.

**Supervisor** See executive routine.

**Swapping** Storing programs on disk and then transferring these programs into main storage as and when they are needed. The technique is used to process large programs or several programs with limited memory.

**Symbolic code** Computer instructions written with the use of English words and/or mathematical symbols instead of binary machine code. Symbolic code is also called source code or source program.

**Symbolic language** A programming language that allows computer instructions to be written with the use of English words and/or mathematical symbols instead of binary machine code.

**Synchronous** The time between successive bits, bytes or events is constant. All equipments connected together in such a system work in step strictly controlled by a clock.

**Syntax** The set of rules of a programming language that define the pattern or structure of the word order and punctuation of an instruction. It is analogous to rules of grammar in English language.

**Syntax errors** Errors in computer programs that typically involve incorrect punctuation, incorrect word sequence, undefined terms, or misuse of terms. These errors are automatically detected and pointed out by language processors.

**System** A group of integrated parts (people, methods, machines, and materials) that have the common purpose of achieving some objective(s).

**System analysis** A detailed step-by-step investigation of related procedures to see what must be done and the best way of doing it.

**System analyst** The individual responsible for planning a computer data processing system. He utilizes tools such as flowcharts, decision tables, program design language, etc. These plans are then passed to the computer programmer.

**System Commands** Commands used to communicate with the operating system of the computer.

**System design** See design phase. Pertains to a detailed description of the new system that has been selected during study phase.

**System flowchart** A flowchart that uses predefined symbols to describe data flow in a system.



**Systems programmer** A person who prepares system programs, that are part of an operating system, designed to simplify the use of the computer.

**Systems software** A set of one or more programs designed to control the operation of a computer system. They are general programs written to assist humans in the use of computer system and for making the operation of the computer system more effective and efficient.

**Tape density** The amount of data that can be placed over a given length of tape. The density is usually expressed in bytes or characters per inch. Typical densities range from 500 bytes per inch to 6250 bytes per inch.

**Teleconferencing** A system in which persons sitting at CRT screens see and talk to each other via a computer-communications network. It saves travelling cost and the valuable time of executives that would otherwise be lost by route.

**Telecommunications** Transmission of data between computer system and/or terminals at different locations through telephone facilities.

**Teletypewriter terminal** A terminal that has a keyboard and a printing device.

**Teletypewriter** An inexpensive input and output device that uses paper tape and is often used as a component of a small computer system.

**Template** A plastic or metal guide used to trace flowcharting symbols.

**Terminal** An input/output device which allows a user to communicate directly with a computer system.

**Terminal symbol** A symbol used in flowcharts to indicate the beginning (START), ending (STOP), and pauses (HALT) in the program logic flow.

**Testing** The process of making sure that the program performs the intended task.

**Thermal printer** A printing device that utilizes paper that is sensitive to heat.

**Third generation computer** Computers built between 1964 and 1975 that used integrated circuits in CPU, high speed magnetic core main memories, powerful high level languages and saw the advent of time sharing operating system.

**Throughput** The total amount of useful processing carried out by a computer system within a given time period. It is a measure of the efficiency of a computer system.

**Time division multiplexing** A method of sharing a communication channel in which the total time available in the channel is divided between several users and each user of the channel is allotted a time slice during which he may transmit a message. The channel capacity is fully utilized by interleaving a number of data streams belonging to different users into one data stream.

**Time log** A log documentation automatically maintained by many computer systems that describe in detail how the computer system was used during the day.

**Time sharing** Refers to the allocation of computer resources in a time-dependent fashion to several programs simultaneously. It facilitates a large number of users to simultaneously use a computer for processing their jobs which may be of different nature.

**Top-down approach** A disciplined approach to system design or program design in which top-level functions are decomposed into a hierarchy of understandable lower-level modules for better management and easy handling.

**Tracing routines** Routines to aid the programmer in following the logic of a program during execution of the program.

**Track** In case of magnetic disk storage, one of many circular concentric rings used for storing data. In case of magnetic tape storage, a horizontal strip, or channel, running the full length of the tape and used for recording data.

**Transaction file** A file in which current data are stored for subsequent processing usually in combination with a master file. Also called a detail file.

**Transistor** A controlled electronic switch fabricated using a semiconductor. It is extensively used in the design of various electronic equipments.

**Transponder** A device mounted on a communication satellite which receives, amplifies and retransmits signals from earth stations.

**Truth table** A table which gives the output values for the various input combinations in case of a logical expression.

**Turnaround time** The elapsed time between the submission of a job to a computer system and getting its output.



**Turnkey systems** Special purpose systems in which a single vendor supplies not only the machine itself, but also all of the needed software such as that to handle general ledger, accounts payable receivable, payroll, inventory and so on.

**Unbundled** A term which means that the application software is priced separately from the computer hardware.

**Unconditional transfer** A program instruction that causes the program control to flow out of normal sequence unconditionally.

**Unit record machine** An electromechanical device used to process data punched into cards. This name came about because it is possible to place one record on one card. Each card containing data is called a unit record.

**UNIVAC** An acronym for Universal Automatic Computer. It was the first commercially available computer.

**Universal Product Code (UPC)** A universal or standardized optical bar code which normally appears on retail packages and which is read by a laser beam scanner.

**UNIX** A popular operating system for 16-bit mini and micro computers that was designed by Bell Telephone Laboratories, USA.

**Updating** The process of keeping master files accurate and up-to-date by processing it with the latest transactions contained in the transaction file.

**User** Any individual who supplies input data to, or uses information generated by, a computer based system.

**Utilities** Programs used to perform some frequently required process in the operation of a computer system-e.g., sorting, merging, transferring data files from one device to another, etc.

**Validation** The process of making sure that forms and documents from a particular transaction are correct.

**Value-added carrier** A communications company that leases networks from a common carrier and combines messages from customers into packets for point-to-point transmission.

**Variable name** In a program, the name assigned to a data field that can assume any of a given set of values.

**Verification** The process by which data contained on cards, tape, or disk is checked against the original document to make sure that no errors occurred in the data conversion.

process. It is the second step of a data entry process.

**Video display unit (VDU)** An I/O device that consists of a television like screen for displaying outputs and a keyboard for entering inputs.

**Virtual storage** The capability to use online secondary storage devices and specialized software to divide programs into smaller segments for transmission to and from internal storage in order to significantly increase the effective size of the available internal storage.

**Virtual system** An extension of virtual storage concept. With virtual systems, users can share all of the computer resources as if each user had his own computer.

**VLSI (Very Large Scale Integration)** An electronic circuit with about 10,000 transistors fabricated in a single silicon chip.

**Voiceband** Data communications system that handles moderate volumes of data, typically from 300 to 9600 bauds. Phone lines that we use to talk to other people is an example.

**Voice recognition unit** An input device used with voice recognition systems that converts spoken words into binary data suitable for input to the system.

**Voice response unit** An output device that uses words or messages recorded on a magnetic medium to produce audio response.

**Volatile storage** A storage medium that loses its contents in the event of power failure or when power is switched off.

**Wide area network (WAN)** A digital communication system which interconnects different sites, computer installations and user terminals, and may also enable LANs to communicate with each other. This type of network may be developed to operate nationwide or worldwide and the transmission medium used are normally public systems such as telephone lines, microwave and satellite links.

**Winchester disk** Medium-sized, non-interchangeable metal disks permanently housed in sealed, contamination-free containers. Read/write heads are built-in with the disk. These disks are used with small, mini, and personal computers.

**Word** A group of bits or characters considered as an entity and capable of being stored in one storage location. Also fixed-size storage areas that form the primary memory of a computer system.



of characters (equal to its word-length in bytes). For such a storage device, storage space is always allocated in multiples of word-length. Its another name is fixed word-length storage.

**Word length** A measure of the size of a word, usually specified in units such as characters or bits. Each location of a computer system can store a fixed number of characters or bits called its word length.

**Word processing** The use of computers to create, view, edit, format, store, retrieve, and print text materials for human

communication.

**Zone area** The 0, 11, and 12 rows on the Hollerith card that are used in combination with digit area punches to indicate letters and some special symbols. Zone areas are also used on other punched-card codes and with internal computer codes.

**Zoned decimal** Any numeric character coded in the Extended Binary Coded Decimal Interchange Code (EBCDIC). Each decimal digit occupies one byte of storage.



















